EXPLORING EVENT LOG ANALYSIS WITH MINIMUM APRIORI INFORMATION

by

Adetokunbo Makanju

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
April 2012

© Copyright by Adetokunbo Makanju, 2012

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "EXPLORING EVENT LOG ANALYSIS WITH MINIMUM APRIORI INFORMATION" by Adetokunbo Makanju in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Dated: April 2, 2012

External Examiner:

Research Supervisors:

Examining Committee:

Departmental Representative:

DALHOUSIE UNIVERSITY

DATE: April 2, 2012

AUTHOR: Adetokunbo Makanju

TITLE: EXPLORING EVENT LOG ANALYSIS WITH MINIMUM APRIORI INFORMATION

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: Ph.D. CONVOCATION: May YEAR: 2012

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions. I understand that my thesis will be electronically available to the public.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than brief excerpts requiring only proper acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

Signature of Author

Table of Contents

List of Tables . . . . **viii**

List of Figures . . . . **xii**

Abstract . . . . **xxiv**

List of Abbreviations and Symbols Used . . . . **xxv**

Acknowledgements . . . . **xxviii**

Chapter 1 Introduction **1**

1.1 Definitions 3

1.2 Autonomic Computing and Fault Management 5

1.3 Motivation and Objectives 9

1.4 Literary Contributions 13

1.5 Organization of the Thesis 14

Chapter 2 Alert Detection in System Logs **16**

2.1 Related and Previous Methods 17

2.2 Limitations of Current Methods 22

2.3 The Proposed Anomaly Detection Framework 23
2.3.1 Event Processing 24
2.3.2 Anomaly Detection 25

Chapter 3 Message Type Extraction **27**

3.1 Background and Motivation 29

3.2 Methodology 31
3.2.1 Simple Logfile Clustering Tool (SLCT) and Loghound 31
3.2.2 Teiresias 35

3.3 The Proposed technique: IPLoM Algorithm 35
3.3.1 Step 1: Partition by event size. 37
3.3.2 Step 2: Partition by token position. 39
3.3.3 Step 3: Partition by search for bijection 41

3.3.4 Step 4: Discover message type descriptions from each partition. 46
3.3.5 Algorithm Parameters 47

3.4 Evaluation and Results 48
3.4.1 Experimental Setting 48
3.4.2 Baseline Experiments 51
3.4.3 Absolute Support Values 56
3.4.4 Percentage-based Support Values 59
3.4.5 Parameter Sensitivity Analysis 59
3.4.6 Complexity 64

3.5 Analysis of Performance Limitations 65
3.5.1 Insufficient Information in Data 65
3.5.2 Ambiguous Token Delimiters 66
3.5.3 Clusters with events of variable size 67

3.6 Discussion of Results 68

Chapter 4 Applications of Message Types in Event Log Management 74

4.1 Visualization 75
4.1.1 Methodology 76
4.1.2 Results 81
4.1.3 Discussion 83

4.2 Message Type Transformation 83

4.3 Storage and Retrieval 84
4.3.1 Methodology 84
4.3.2 Results 87

4.4 Building Computational Models 88
4.4.1 Methodology 89
4.4.2 Result 92

4.5 Discussion 93

Chapter 5 Spatio-Temporal Decomposition, Correlated Message Types and System State 100

5.1 Related Work 101

5.2 Methodology 103
5.2.1 Information Content Clustering (ICC) 109
5.2.2 Evaluations 110

5.3 Results 114
5.3.1 Internal Measures 114
5.3.2 Ground Truth 114
5.3.3 Discussion on Hypotheses 118

5.4 Discussion of Results 118

Chapter 6 Spatio-Temporal Alert Detection in HPC Logs 124

6.1 Related Work 125

6.2 Methodology 125
6.2.1 Spatio-Temporal Decomposition of log events 126
6.2.2 Clustering of Spatio-Temporal partitions 127
6.2.3 Identification of Anomalous Clusters 133

6.3 Cluster Separation 133
6.3.1 Alert Characteristics 133
6.3.2 Alert Cluster Assumptions 136

6.4 Identification Rules 137
6.4.1 Rule Definitions 138
6.4.2 Evaluations 141

6.5 Results 143

6.6 Anomaly Detection using C5.0 145
6.6.1 Discussion of C5.0 Results 150

6.7 Final Discussion of Results 151

Chapter 7 Spatio-Temporal Alert Detection in Non-HPC Logs . . 153

7.1 Methodology 153

7.2 Results 157

7.3 Discussion 160

Chapter 8 Putting it All Together: Online Alert Detection through Interactive Learning of Alert Signatures 171

8.1 Methodology 172
8.1.1 Anomaly Detection 174
8.1.2 Signature Generation 175
8.1.3 Visualization, Feedback and Cluster Refinement 178

8.2 Complexity 180

8.3 Evaluations 181

8.4 Results 183

8.5 Discussion 187

Chapter 9 Conclusion 193

9.1 Contributions 195

9.2 Future Research Directions 197

Bibliography 202

Appendix A 210

A.1 Tool Documentation 210
A.1.1 Full List of SLCT Options 210
A.1.2 Full List of Loghound Options 213
A.1.3 Full List of C5.0 Options 215
A.1.4 Full List of Apriori Options 216

Appendix B 218

B.1 Comparing Entropy-Based Approaches in Online Alert Detection . . 218
B.1.1 Results 219
B.1.2 Conclusion 222

Appendix C 231

C.1 Further Results And Figures 231
C.1.1 HPC Node Circos Plots 231
C.1.2 Other HPC Dissimilar Node Circos Plots 244
C.1.3 Detailed Results for C5.0 Evaluations 262

Appendix D 267

D.1 Analysis of Variance (ANOVA) Tables 267
D.1.1 IPLoM Precision Performance 267
D.1.2 IPLoM Recall Performance 270
D.1.3 IPLoM F-Measure Performance 272
D.1.4 STAD and C5.0 Rule-set ANOVA Results 275
D.1.5 Online detection ANOVA Results 277

Appendix E 281

E.1 Comprehensive List of Literary Contributions of the Thesis 281

List of Tables

1.1 Placing the framework within the context of autonomic computing . . . 13

3.1 Log Data Statistics . . . 49

3.2 Algorithm Parameters . . . 51

3.3 Log Data Event Size Statistics . . . 54

3.4 Algorithm performance based on cluster event size . . . 55

3.5 Algorithm performance based on cluster instance frequency ($LH \rightarrow$ *Loghound*) . . . 55

3.6 CPU Time in Minutes . . . 57

3.7 Maximum Virtual Memory Consumption in KBs . . . 57

3.8 Maximum Resident Memory Consumption in KBs . . . 58

3.9 Average F-Measure performance of algorithms using absolute support values . . . 58

3.10 Parameter Value Ranges Used for Sensitivity Analysis . . . 64

3.11 Standard Deviation over F-Measure results for parameter values 65

4.1 Dataset Summary . . . 78

4.2 Message Type Summary . . . 80

4.3 Data Attributes Used in TreeML Files . . . 81

4.4 HPC Log Data Statistics . . . 86

4.5 Percentage Reduction in # of terms . . . 88

4.6 Percentage Reduction in File Size . . . 89

4.7 Functional Group Data Statistics . . . 92

5.1 Statistics of Message Type Distribution across Nodehours for HPC Data . . . 108

5.2 Cluster Statistics . . . 122

5.3 Average FWithin and *Distance* for Nodehour Clusters 122

5.4 Alert Nodehour Cluster Conceptual Purity 123

5.5 Cluster Measurements for a Random Single Node 123

6.1 Sample Confusion Matrix 130

6.2 Similarity Status of Node Functional Groupings used in Evaluations 143

6.3 ANOVA Test Summary 145

6.4 C5.0 Parameter Settings 149

7.1 Non-HPC Log Data Statistics 154

7.2 Distributed-Log Dataset Statistics 155

7.3 Cloud-Log Dataset Statistics 156

7.4 Average FWithin, Distance and Conceptual Purity for Nodehour Clusters from the Distributed Log 158

7.5 Average FWithin, Distance and Conceptual Purity for Nodehour Clusters from the Distributed Log 158

8.1 ANOVA Test Summary 184

8.2 Number of Signatures Per Alert Type 186

8.3 Number of Signatures Per Alert Type 186

8.4 Number of Signatures Per Alert Type 187

9.1 Placing the framework within the context of autonomic computing 194

B.1 Average performance across all datasets for *NIPlus* and *NIUniq* using a window size of 30 days 225

B.2 Average performance across all datasets for *NIUniq* using window sizes of 30, 60 and 90 days 225

D.1 Anova Results Comparing Precision Performance on HPC Log. 267

D.2 Anova Results Comparing Precision Performance on Syslog Log. 268

D.3 Anova Results Comparing Precision Performance on Windows Log. 268

D.4 Anova Results Comparing Precision Performance on Access Log. 268

D.5 Anova Results Comparing Precision Performance on Error Log. 269

D.6 Anova Results Comparing Precision Performance on System Log. 269

D.7 Anova Results Comparing Precision Performance on Rewrite Log. 269

D.8 Anova Results Comparing Recall Performance on HPC Log. . . 270

D.9 Anova Results Comparing Recall Performance on Syslog Log. . 270

D.10 Anova Results Comparing Recall Performance on Windows Log. 271

D.11 Anova Results Comparing Recall Performance on Access Log. . 271

D.12 Anova Results Comparing Recall Performance on Error Log. . . 271

D.13 Anova Results Comparing Recall Performance on System Log. . 272

D.14 Anova Results Comparing Recall Performance on Rewrite Log. 272

D.15 Anova Results Comparing F-Measure Performance on HPC Log. 273

D.16 Anova Results Comparing F-Measure Performance on Syslog Log. 273

D.17 Anova Results Comparing F-Measure Performance on Windows Log. 273

D.18 Anova Results Comparing F-Measure Performance on Access Log. 274

D.19 Anova Results Comparing F-Measure Performance on Error Log. 274

D.20 Anova Results Comparing F-Measure Performance on System Log. 274

D.21 Anova Results Comparing F-Measure Performance on Rewrite Log. 275

D.22 Anova Results Comparing Recall Performance of C5.0 and Rule-Set on Cluster Classification 275

D.23 Anova Results Comparing FPR Performance of C5.0 and Rule-Set on Cluster Classification 276

D.24 Anova Results Comparing Recall Performance of Rule-Set Cluster Classifier when parameters are set automatically and best case when parameters are set manually. 276

D.25 Anova Results Comparing FPR Performance of Rule-Set Cluster Classifier when parameters are set automatically and best case when parameters are set manually. 276

D.26 Anova Results Comparing Recall Performance of *NIPlus* and best case of Rule-Set Cluster Classifier 277

D.27 Anova Results Comparing FPR Performance of *NIPlus* and Rule-Set Cluster Classifier when parameters are set automatically. . 277

D.28 Anova Results Comparing Recall Performance with different training file split points on the HPC Logs. 278

D.29 Anova Results Comparing FPR Performance with different training file split points on the HPC Logs. 278

D.30 Anova Results Comparing DR Performance with different training file split points on the Distributed Logs. 279

D.31 Anova Results Comparing FPR Performance with different training file split points on the Distributed Logs. 279

D.32 Anova Results Comparing DR Performance with different training file split points on the Cloud Logs. 280

D.33 Anova Results Comparing FPR Performance with different training file split points on the Cloud Logs. 280

List of Figures

1.1 An system log file example. Each line represents an event. . . 4

1.2 Sample event broken down into its constituent fields. 4

1.3 Overview of the hybrid interactive learning framework. 12

2.1 Overview of the STAD framework. 24

3.1 An example system log event. 29

3.2 Sample clusters generated by SLCT and Loghound 34

3.3 Overview of IPLoM processing steps. 37

3.4 IPLoM Step-1: Partition by event size. Separates the messages into partitions based on the their event sizes. 38

3.5 IPLoM Step-2: Partition by token position. Selects the token position with the least number of unique values, token position 2 in this example. Then, it separates the messages into partitions based on unique token values, i.e. "plb" and "address:", in the token position. 40

3.6 Example messages illustrating 1-M, M-1 and M-M relationships. 43

3.7 IPLoM Step-3: Partition by search for bijection. 44

3.8 Deciding on how to treat 1-M and M-1 relationships. This procedure is implemented in the Get_Rank_Position function. . 46

3.9 IPLoM Step-4: Discover message type descriptions. If the cardinality of the unique token values in a token position is equal to 1, then that token position is represented by that token value in the template, else the token position is represented with an "*". 47

3.10 Comparing algorithm IR performances at lowest support values. 53

3.11 Comparing Recall performance of IPLoM, Loghound and SLCT using support values. 60

3.11 Comparing Recall performance of IPLoM, Loghound and SLCT using support values. 61

3.12 Comparing Precision performance of IPLoM, Loghound and SLCT using support values. 61

3.12 Comparing Precision performance of IPLoM, Loghound and SLCT using support values. 62

3.13 Comparing F-Measure performance of IPLoM, Loghound and SLCT support values. 62

3.13 Comparing F-Measure performance of IPLoM, Loghound and SLCT support values. 63

3.14 F-Measure performance of IPLoM, Loghound and SLCT against FST values in the range 0% - 100%. 0% support values for SLCT and Loghound are equivalent to using an absolute support value of 2. 70

3.14 F-Measure performance of IPLoM, Loghound and SLCT against FST values in the range 0% - 100%. 0% support values for SLCT and Loghound are equivalent to using an absolute support value of 2. 71

3.15 Runtime Cost of IPLoM on the BGL data. Each Bgl$x$ dataset contains $x * 100,000$ events. 71

3.16 Example: Insufficient Information in Data. 72

3.17 Comparing F-Measure performance of IPLoM, Loghound and SLCT before insertion of counter-examples, after insertion of counter examples and evaluating the accuracy of cluster partitioning before the insertion of counter examples. 72

3.17 Comparing F-Measure performance of IPLoM, Loghound and SLCT before insertion of counter-examples, after insertion of counter examples and evaluating the accuracy of cluster partitioning before the insertion of counter examples. 73

4.1 Treemap visualization of hierarchical data using the slice and dice algorithm. 78

4.2 General outline of a TreeML file 79

4.3 An Overview of the LogView Interface. 82

4.10 Proposed Event Storage and Retrieval Procedure. 85

4.4 Figure showing the treemaps produced by LogView. (a)SSH (b) IMAP (c) POP3 (d)HTTP. 94

4.5 Analysis Using LogView with the SSH service (a) Search (b) Filtering (c) Selection. 95

4.6 Zooming and Panning with the SSH service.(a) Zoomout (b) Zoomin and Pan-Left. 95

4.7 **Phrasal Message Type Transformation:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template which matches the message in the first box on the right. Each phrase consisting of constant tokens is replaced with a unique token. In the final box on the right, *XX* represents an ordinal number assigned to the message type and hence will change for different message types. 96

4.8 **Message Type Transformation with variables:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template which matches the message in the first box on the right. In the final box on the right, *XX* represents an ordinal number assigned to the message type and hence will change for different message types. 96

4.9 **Full Message Type Transformation:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template which matches the message in the first box on the right. This method is identical to the message type transformation with variables except that the variable values are discarded in the final transformation. In the final box on the right, *XX* represents an ordinal number assigned to the message type and hence will change for different message types. 97

4.11 Process flow for experiments. The baseline for performance had the raw data input into Nodeinfo without preprocessing using IPLOM and MTTs. 97

4.12 Precision-Recall plots for the BGL node categories using the original Nodeinfo framework (NI) and modified framework (NIPlus) obtained by transforming the raw log data using MTT-3. 98

4.13 Precision-Recall plots for the Liberty node categories using the original Nodeinfo framework (NI) and modified framework (NIPlus) obtained by transforming the raw log data using MTT-3. 98

4.14 Precision-Recall plots for the Spirit node categories using the original Nodeinfo framework (NI) and modified framework (NI-Plus) obtained by transforming the raw log data using MTT-3. 99

4.15 Precision-Recall plots for the Tbird node categories using the original Nodeinfo framework (NI) and modified framework (NI-Plus) obtained by transforming the raw log data using MTT-3. 99

5.1 Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the BGL node functionality categories. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on ICS in the plot. 105

5.2 Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the Liberty node functionality categories. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on ICS in the plot. 106

5.3 Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the Spirit node functionality categories. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on ICS in the plot. 107

5.4 Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the Tbird node functionality categories. The plot differentiates between alert nodehours and normal nodehours; the Tbird-SM category has no alert nodehours. Nodehours are sorted based on ICS in the plot. 108

5.5 Two example nodehour clusters formed using Algorithm 5. The enclosed boxes represent nodehours which belong to the cluster and the bulleted lists represent the set of unique message types which appear in the nodehour. In each bulleted list the message type highlighted in *red* represents the message type with maximum entropy which is common to all nodehours in a cluster based on Algorithm 5. 112

5.6 Stacked Bar Graphs for BGL node functionality categories. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). Only clusters which contain 3 or more nodehours are plotted. 115

5.7 Stacked Bar Graphs for Liberty node functionality categories. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). Only clusters which contain 3 or more nodehours are plotted. 116

5.8 Stacked Bar Graphs for Spirit node functionality categories. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). Only clusters which contain 3 or more nodehours are plotted. 117

5.9 Stacked Bar Graphs for Tbird node functionality categories. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represents the distribution of alert nodehours (red) to normal nodehours (blue). Only clusters which contain 3 or more nodehours are plotted. 118

6.1 Stacked Bar Graphs for a single node from node functionality categories which are considered dissimilar. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). All clusters which contain at least one alert nodehour are shown. 129

6.2 Simple 2×2 Circos plot example. 131

6.3 Example of a more complex Circos plot. 132

6.4 **Bursty Property:** This graph shows the size of the events produced by a single node from the Liberty HPC at hourly intervals over a 24 hour period. 134

6.5 **Endemic Property:** The graph shows localized sporadic activity. 135

6.6 **Epidemic Property:** The graph shows sporadic activity which affects a relatively large number of nodes. 136

6.7 **Near-Periodic Property:** The graph shows very frequent activity compared to either the endemic or epidemic types and has an almost periodic rate of occurrence. 137

6.8 **Liberty-Compute Alert Clusters:** The graph shows a scatter plot of the alert clusters derived from the Liberty HPC log versus their inverse node frequency scores. The clusters are sorted based on their average inverse node frequency scores. . 139

6.9 **Spirit Admin Clusters:** This graph shows a bar chart of clusters with a *norm_per_cnt* > 0.1 from the Admin functional node category of the Spirit HPC log. Cluster 16 is the only cluster which is known to contain alerts. 141

6.10 This figure shows recall and false positive rates for evaluations of STAD on the datasets listed in Table 4.7. 144

6.11 This graph shows the best case results for the experiments in which the *med_per* parameter is set manually and the results in which its value is set automatically. In addition, it shows the false positive rate achieved by *NIPlus* for a recall rate similar to the autoset experiments. 146

6.12 Comparing the Recall and FP rates for the default performance of C5.0 and the anomaly detection rule set. 150

7.1 Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the distributed log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot. 161

7.1 Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the distributed log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot. 162

7.1 Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the distributed log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot. 163

7.2 Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the cloud log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot. 164

7.2 Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the cloud log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot. 165

7.3 Stacked Bar Graphs for the application logs from the distributed log. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). All clusters which contain at least one alert nodehour are shown. 166

7.4 Stacked Bar Graphs for the application logs from the cloud log. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). All clusters which contain at least one alert nodehour are shown. 167

7.5 Recall and FP rates for STAD on the application categories of the distributed log. 168

7.6 Recall and FP rates for STAD on the application categories of the cloud log. 168

7.7 Scatter-Plot of spatio-temporal partitions (x-axis) vs. ICSs (y-axis) for the Sendmail and Secure application logs. For each application, the temporal granularity level of the plot is “Hours” for the graph on the left while “Minutes” are used for the graph on the right. 169

7.7 Scatter-Plot of spatio-temporal partitions (x-axis) vs. ICSs (y-axis) for the Sendmail and Secure application logs. For each application, the temporal granularity level of the plot is “Hours” for the graph on the left while “Minutes” are used for the graph on the right. 170

8.1 Framework for Automatic Signature Generation from System Logs. This figure highlights the phases involved in the proposed framework for automatic alert signature generation using the contents of the system’s historical log files. 174

8.2 Spatio-Temporal Alert Detection (STAD). This figure shows the steps in the STAD alert detection mechanism. The mechanism results in the classification of the contents of log into *normal* and *anomalous* categories. 175

8.3 Signature Creation Mechanism. This figure shows the steps in the signature creation phase of the framework. 177

8.4 Online Alert Detection. This shows how the alert signatures produced by the system can be used for online alert detection on a production system. 178

8.5 The figure above shows three examples of spatio-temporal partition clusters which maybe formed during analysis. For this example, let us assume that Clusters A and C, have been identified correctly as *normal* while Cluster B has been wrongfully identified as *anomalous* due to the fact that it contains only one spatio-temporal partition. 180

8.6 The figure shows the single spatio-temporal partition from Cluster B in Fig. 8.5. It depicts a possible scenario of breaking up the spatio-temporal partition into smaller and temporally uneven spatio-temporal units. 181

8.7 Detection rates for the different HPC, Distributed and Cloud logs. For the HPC log, the files are numbered in the same order as shown in Table 4.7 189

8.8 Detection rates for the different HPC logs, showing variations in performance for different sizes of training files 190

8.9 Boxplots showing distributions for signature length, i.e. the number of message types which define a signature. 191

8.10 Boxplots showing distributions for signature support levels . . 192

B.1 Precision-Recall plots for the BGL node categories. Each plot compares the result of using the original nodeinfo framework with message types incorporated (NIPlus) and nodeinfo based on Eq. B.1 (NIUniq). A window size of 30 was used for both methods. 219

B.2 Precision-Recall plots for the Liberty node categories. Each plot compares the result of using the original nodeinfo framework with message types incorporated (NIPlus) and nodeinfo based on Eq. B.1 (NIUniq). A window size of 30 was used for both methods. 220

B.3 Precision-Recall plots for the Spirit node categories. Each plot compares the result of using the original nodeinfo framework with message types incorporated (NIPlus) and nodeinfo based on Eq. B.1 (NIUniq). A window size of 30 was used for both methods. 221

B.4 Precision-Recall plots for the Tbird node categories. Each plot compares the result of using the original nodeinfo framework with message types incorporated (NIPlus) and nodeinfo based on Eq. B.1 (NIUniq). A window size of 30 was used for both methods. 222

B.5 Precision-Recall plots for the BGL node categories. Each plot compares the result of using the nodeinfo based on Eq. B.1 (NIUniq) while varying the window size to 30, 60 and 90 days. 223

B.6 Precision-Recall plots for the Liberty node categories. Each plot compares the result of using the nodeinfo based on Eq. B.1 (NIUniq) while varying the window size to 30, 60 and 90 days. 224

B.7 Precision-Recall plots for the Spirit node categories. Each plot compares the result of using the nodeinfo based on Eq. B.1 (NIUniq) while varying the window size to 30, 60 and 90 days. 225

B.8 Precision-Recall plots for the Tbird node categories. Each plot compares the result of using the nodeinfo based on Eq. B.1 (NIUniq) while varying the window size to 30, 60 and 90 days. 226

B.9 The bar chart compares the FPR achieved at the maximum F-Measure score and at 90% Recall for the experiment trials on the BGL node categories shown in Figs. B.1, B.5. 227

B.10 The bar chart compares the FPR achieved at the maximum F-Measure score and at 90% Recall for the experiment trials on the Liberty node categories shown in Figs. B.2, B.6. 228

B.11 The bar chart compares the FPR achieved at the maximum F-Measure score and at 90% Recall for the experiment trials on the Spirit node categories shown in Figs. B.3, B.7. 229

B.12 The bar chart compares the FPR achieved at the maximum F-Measure score and at 90% Recall for the experiment trials on the Tbird node categories shown in Figs. B.4, B.8. 230

C.1 Circos Plots for nodes for the BGL node categories. Each plot shows the plot for clustering as performed on all nodehours in each category. 232

C.1 Circos Plots for the BGL node categories. Each plot shows the plot for clustering as performed on all nodehours in each category.233

C.1 Circos Plots for the BGL node categories. Each plot shows the plot for clustering as performed on all nodehours in each category. 234

C.1 Circos Plots for the BGL node categories. Each plot shows the plot for clustering as performed on all nodehours in each category. 235

C.2 Circos Plots for the Liberty node categories. The Liberty-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Liberty-Admin and Liberty-Other categories. 236

C.2 Circos Plots for the Liberty node categories. The Liberty-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Liberty-Admin and Liberty-Other categories. 237

C.2 Circos Plots for the Liberty node categories. The Liberty-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Liberty-Admin and Liberty-Other categories. 238

C.3 Circos Plots for the Spirit node categories. The Spirit-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Spirit-Admin and Spirit-Other categories. 239

C.3 Circos Plots for the Spirit node categories. The Spirit-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Spirit-Admin and Spirit-Other categories. 240

C.3 Circos Plots for the Spirit node categories. The Spirit-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Spirit-Admin and Spirit-Other categories. 241

C.4 Circos Plots for the Tbird node categories. The Tbird-Compute and Tbird-Other plots show plots for clustering as performed on all nodehours in each category, while only clusters formed from a single node are shown for Tbird-Admin category. . . . 242

C.4 Circos Plots for the Tbird node categories. The Tbird-Compute and Tbird-Other plots show plots for clustering as performed on all nodehours in each category, while only clusters formed from a single node are shown for Tbird-Admin category. . . . 243

C.4 Circos Plots for the Tbird node categories. The Tbird-Compute and Tbird-Other plots show plots for clustering as performed on all nodehours in each category, while only clusters formed from a single node are shown for Tbird-Admin category. . . . 244

C.5 Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category. 245

C.5 Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category. 246

C.5 Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category. 247

C.5 Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category. 248

C.5 Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category. 249

C.5 Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category. 250

C.6 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 251

C.6 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 252

C.6 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 253

C.6 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 254

C.6 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 255

C.7 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 256

C.7 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 257

C.7 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 258

C.7 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 259

C.7 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 260

C.7 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 261

C.7 Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category. . . 262

C.8 Recall and FPR scores for the classification of anomalous and normal clusters for the BGL node categories using C5.0 classification rules. 263

C.9 Recall and FPR scores for the classification of anomalous and normal clusters for the Liberty node categories using C5.0 classification rules. 264

C.10 Recall and FPR scores for the classification of anomalous and normal clusters for the Spirit node categories using C5.0 classification rules. 265

C.11 Recall and FPR scores for the classification of anomalous and normal clusters for the Tbird node categories using C5.0 classification rules. 266

Abstract

The continued increase in the size and complexity of modern computer systems has led to a commensurate increase in the size of their logs. System logs are an invaluable resource to systems administrators during fault resolution. Fault resolution is a time-consuming and knowledge intensive process. A lot of the time spent in fault resolution is spent sifting through large volumes of information, which includes event logs, to find the root cause of the problem. Therefore, the ability to analyze log files automatically and accurately will lead to significant savings in the time and cost of downtime events for any organization. The automatic analysis and search of system logs for fault symptoms, otherwise called *alerts*, is the primary motivation for the work carried out in this thesis.

The proposed log alert detection scheme is a hybrid framework, which incorporates anomaly detection and signature generation to accomplish its goal. Unlike previous work, minimum apriori knowledge of the system being analyzed is assumed. This assumption enhances the platform portability of the framework. The anomaly detection component works in a bottom-up manner on the contents of historical system log data to detect regions of the log, which contain anomalous (alert) behaviour. The identified anomalous regions are then passed to the signature generation component, which mines them for patterns. Consequently, future occurrences of the underlying *alert* in the anomalous log region, can be detected on a production system using the discovered pattern. The combination of anomaly detection and signature generation, which is novel when compared to previous work, ensures that a framework which is accurate while still being able to detect new and unknown alerts is attained.

Evaluations of the framework involved testing it on log data for High Performance Cluster (HPC), distributed and cloud systems. These systems provide a good range for the types of computer systems used in the real world today. The results indicate that the system that can generate signatures for detecting alerts, which can achieve a Recall rate of approximately 83% and a false positive rate of approximately 0%, on average.

List of Abbreviations and Symbols Used

AI	Artificial Intelligence
ANOVA	Analysis of Variance
CBA	Classification-Based Association
CBE	Common Base Event
CBR	Case-Based Reasoning
CGR	Cluster Goodness Ratio
CGT	Cluster Goodness Ratio Threshold
CMAR	Classification based on Multiple Association Rules
CPAR	Classification based on Predictive Association Rules
CPU	Central Processing Unit
ERN	Event Relationship Network
FN	False Negative
FP	False Positive
FPR	False Positive Rate
FSA	Finite State Auotmata
FST	File Support Threshold
HPC	High Performance Cluster
HPC	High Performance Clusters
HTTP	Hyper Text Transfer Protocol
ICC	Information Content Clustering
ICS	Information Content Score
IMAP	Internet Message Access Protocol
IPLoM	Iterative Partitioning Log Mining

LB	Lower Bound
LSA	Latent Semantic Analysis
LSI	Latent Semantic Indexing
MTI	Message Type Indexing
MTT	Message Type Transformations
OS	Operating System
PAM	Pluggable Authentication Module
PCA	Principal Component Analysis
PHP	Hypertext Processor
POP3	Post Office Protocol V3
PSR	Partition Support Ratio
PST	Partition Support Threshold
RFC	Request for Comment
SLCT	Simple Logfile Clustering Tool
SSH	Secure Shell
STAD	Spatio-Temporal Alert Detection
SVD	Singular Value Decomposition
TCP/IP	Transmission Control Protocol/Internet Protocol
TN	True Negative
TP	True Positive
UB	Upper Bound

XML	Extensible Markup Language

Acknowledgements

Working on this dissertation has perhaps been the most involved endeavour which I have undertaken so far. As it comes that an end I would like to thank the many people who have helped to make it possible.

To my supervisors Dr. Nur Zincir-Heywood and Dr. Evangelos Milios. I consider myself lucky to have worked under you.

To the members of my supervisory committee, Dr. Malcolm Heywood, Dr. Srinivas Sampalli and the external examiner Dr. Stan Matwin. Thanks for your valuable comments and criticisms.

To Markus Latzel, Steve Stergiopoulos and Raffael Marty. Good data is at the core of any thesis, thanks for giving me access to yours.

To Dr. Wale Babalakin. Words cannot express my gratitude.

To my Dad. Once there was one but now they are two.

To my Mum. It hurts that you are not here to see this. Though you seem so far I know that you are near.

To all my many friends and relations have who have given me love, support and encouragement. It was not in vain

And finally to God. You make all things possible.

Chapter 1

Introduction

Modern computer systems and networks are increasing in size and complexity at an exponential rate. An observation of the progression towards the building of large scale data-centers to support cloud infrastructure and the drive to build bigger and larger High Performance Clusters (HPC) which can perform computations on the ever increasing amounts of data being collected, will confirm this assertion. For the first time all the top 10 HPC (supercomputers) in the world are capable of computing performance in the petaflop (quadrillion) range. The fastest supercomputer in the world, the K Computer located at the RIKEN Advanced Institute of Computational Science (AICS) in Japan is capable of 8 quadrillion calculations per second and has 68,544 nodes with 8 cores each, leading to a total of 548,352 cores [73]. It is fair to say that this trend is bound to continue.

Fault resolution on modern computer systems is already a time-consuming and knowledge intensive process. System administrators typically spend hours sifting through lots of data, firstly to find the root cause of the problem and then plan the resolution. Typically, finding the root cause will involve analyzing system state information and the operational context. Sources of information about activity on computer systems or computer networks include application log files, system activity reports, trouble tickets, co-workers, off-site sources and system activity paths. As the size and complexity of systems continue to increase, so will the size and complexity of these data sources. Therefore, it can only be expected that the tasks required for the management of error and fault conditions on such systems will reach a level of complexity where some degree of automation will be required to keep up with the pace.

The goal of autonomic computing as espoused by IBM's senior vice president of research, Paul Horn, in March 2001 can be defined as the goal of building self-managing computing systems [28]. This is a long-term goal, but in the short-term the design of computing systems which can gather and analyze information about their states automatically, to support decisions made by human administrators needs to be explored. Eventually, this intermediary short-term goal will make the long-term goal achievable[28].

Self-healing in autonomic computing has set a goal for the engineering of systems which are capable of detecting and fixing their own error conditions. With the losses incurred in both man hours and operational cost during downtime events, such systems already have a ready market in the world of information technology. This dissertation contributes towards the goal of building such systems, while focusing only on one of the many sources of information available, i.e. event logs or system logs. These logs are generated by the many components which make up a computer system and consist of several independent lines of text data which contain information that pertains to events which occur within a system. Event logs, which are the largest on-site source of information (by volume) available to system administrators, are a ready and available source of data during down time events. However trends tend to suggest that the task of analyzing these logs is becoming too cumbersome to be carried out manually [68], so the approach proposed by this thesis is timely.

This work proposes a framework which attempts to detect error conditions (alerts) automatically in computer systems that are detectable in system logs. The computer systems can either be on wired or wireless infrastructures, as the assumption here is that logs have been reported by the application layer or generally components running above Layer-2 in the protocol stack. I differentiate here between errors as symptoms of a fault and the actual faults. Faults usually leave traces on systems before and after they occur. These traces manifest themselves in the form of errors (alerts) in the system. The goal here is the automatic identification of these error conditions, thereby reducing troubleshooting time and preventing downtime events altogether. As full automation is difficult and labour intensive, while low level monitoring tools still require a lot of manual input with regard to rule maintenance, the proposed framework aims to be a middle level solution which provides the right amount of automation with minimal manual activity.

Application log files pose a lot of challenges for the task at hand. One is their semi-structured nature and another is their diversity, leading to inconsistency. Another possible challenge is that log files may not capture fully information about failure in all cases. Moreover, application developers cannot anticipate all fault conditions which can occur. Therefore, this framework will employ a mix of data mining and information-theoretic techniques to overcome some of these challenges. While the focus of this research is fault management, the intention is to produce a framework which is general enough to apply to other network management functions. The framework proposed will will extract information from system log files automatically, thus reducing the manual input required to analyze the contents of event logs. The system can analyze its own logs automatically and provide hints to the administrator about possible error conditions. Once error conditions are confirmed, the system can develop signatures for such errors and flag them when they occur in the future, thereby reducing the time and effort exerted by the administrator in log analysis and management.

1.1 Definitions

This section contains a glossary of some of the terminology used in this thesis.

- **Event Log or System Log.** A text-based audit trail of events which occur within the system or application processes on a computer system (Fig. 1.1).

- **Event.** An independent line of text within an event log which details a single occurrence on the system, (Fig. 1.2). An event typically contains not only a *message* but other fields of information such as the *Date*, *Source* and *Tag* as defined in the `syslog` RFC (Request for Comment) [38]. In Fig. 1.2 the first five fields (delimited by whitespace) represent the *Timestamp*, *Host*, *Class*, *Facility* and *Severity* of each event.

- **Token or Term.** A single word delimited by white space within the *message* field of an event. For example in Fig. 1.2, the words *invalid* and *SNA.....0* are tokens in that message.

- **Event Size.** The number of individual tokens in the "message" field of an event. The event in Fig. 1.2 has an event size of 2.

```
2005-06-03-15.42.50.823719 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
2005-06-03-15.42.50.982731 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
2005-06-06-22.41.37.357738 R20-M0-NA-C:J15-U11 RAS KERNEL INFO generating core.3740
2005-06-06-22.41.37.392258 R20-M0-NA-C:J17-U11 RAS KERNEL INFO generating core.3612
2005-06-11-19.20.25.104537 R30-M0-N9-C:J16-U01 RAS KERNEL FATAL data TLB error interrupt
2005-06-11-19.20.25.393590 R30-M0-N9-C:J16-U01 RAS KERNEL FATAL data TLB error interrupt
2005-07-01-17.52.23.557949 R22-M0-NA-C:J05-U01 RAS KERNEL INFO 458720 double-hummer alignment exceptions
2005-07-01-17.52.23.584839 R22-M0-NA-C:J03-U01 RAS KERNEL INFO 458720 double-hummer alignment exceptions
```

Figure 1.1: An system log file example. Each line represents an event.

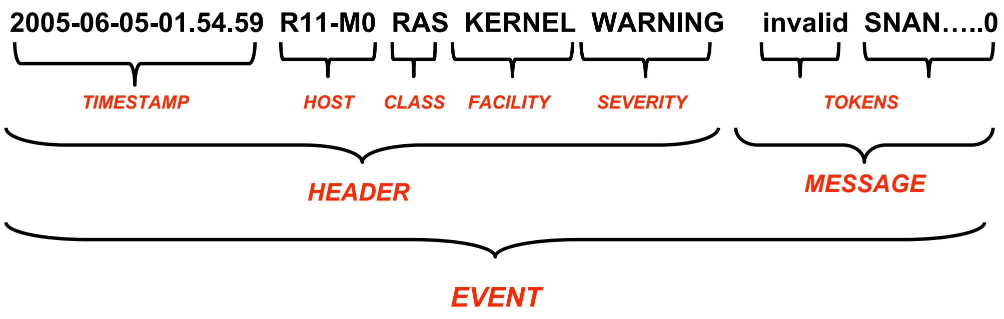


Figure 1.2: Sample event broken down into its constituent fields.

- **Message Type.** These are *message* fields of entries within an event log produced by the same print statement. Non-overlapping consecutive pairs of lines in the log example shown in Fig. 1.1 belong to the same message type.

- **Message Type Description.**A textual template containing wild-cards which represents all members of an event cluster. The messages in the 3rd and 4th lines of Fig. 1.1 have a message type description of *"generating *"*.

- **Constant Token.** A token within the *message* field of an event which is not represented by a wild-card value in its associated message type description. The token *generating* in the 3rd line of Fig. 1.1 is a constant token.

- **Variable Token.** A token within the *message* field of an event which is represented by a wild-card value in its associated message type description. The token *core.3740* in the 3rd line of Fig. 1.1 is a variable token.

1.2 Autonomic Computing and Fault Management

This work can be placed within the context of autonomic computing and fault management systems. First of all, this section discusses where this work falls within the context of the state of the art in autonomic computing and fault management.

Autonomic computing is a term inspired by the workings of the autonomic nervous system. Like the autonomic nervous system which works to control certain bodily functions without the conscious thought of the organism, the aim of autonomic computing is to provide modern computer systems with the ability to manage their internal states for optimal performance with little or no human intervention, i.e. self-managing systems. The four objectives of the self-managing paradigm of autonomic computing are: self-configuration, self-optimization, self-healing and self-protection [28, 13]. These objectives are not mutually exclusive and are interdependent in most cases [13]. For instance to achieve self-protection, self-configuration may be necessary. The objectives are described below.

- **Self-Configuration.** This refers to the ability of the system to choose optimal settings automatically for its configuration parameters to meet the goals of an organization as specified by the human operator. This configuration should not be static; it should have the ability to adapt automatically when changes occur or when the goals set by the human operator change.

- **Self-Optimization.** This refers to the ability of the system to adapt to changes in its state and environment. These changes should help to achieve an optimal usage of resources so as to maintain performance levels or Quality of Service(QoS).

- **Self-Healing.** This refers to the ability of the system to detect, diagnose and recover from fault conditions as they occur on the system.

- **Self-Protection.** This refers to the ability of the system to respond to security vulnerabilities. The system should be able to protect itself pro-actively from malicious activity and benign user actions which could lead to a security breach.

For an autonomic system to achieve these four 'self-' objectives they need to posses four attributes [13]. These are self-awareness, self-situation, self-monitoring and self-adjustment. Self-awareness refers to the system's ability to be informed about its internal state, while self-situation implies the ability to be informed about its external operating conditions. Self-monitoring is the ability to detect changes, while the ability to adapt to these changes is self-adjustment.

Since the introduction of the concept in 2001 by IBM [28], several approaches, techniques and frameworks have been proposed. A full discussion of all of these is beyond the scope of this work but summaries of the state of the art in autonomic computing can be found in Huebscher and McCann [22] and Khalid et al. [29]. Any framework, which aims to achieve self-managing capabilities for a system can be grouped primarily on the basis of its intended objectives and attributes [28, 13].

Furthermore, frameworks can be grouped on the basis of their design approach [16, 29] or the framework type. Two common design approaches are described below.

- **Externalization Approach.** In this approach to designing autonomic systems, the components which enable autonomicity are situated outside of the system being managed. This is the most common approach and is considered more effective [29].

- **Internalization Approach.** In this approach to designing autonomic systems, the components which enable autonomicity are part of the system itself, i.e. internal [29].

On the other hand, framework types include the following [29].

- **Biologically Inspired.** These are autonomic systems which are inspired by or mimic biological systems. A good example of this is the Autonomous Nanotechnology Swarm (ANTS) [13].

- **Large Scale Distributed.** These autonomic systems focus on the management of large scale distributed systems. Large scale distributed systems have to be highly available and scalable; doing this requires careful configuration and organization. Large scale autonomic systems attempt to carry out this configuration and organization automatically. Examples include IBM's SMART and Microsoft's AutoAdmin [29].

- **Agent-Based.** Agent-based autonomic systems use a decentralized approach to achieve autonomicity. Several independent agents work alone and in concert with other agents to achieve autonomicity. The application monitoring in the Autonomia system of the University of Arizona [29] is an example of an autonomic system which is based on an agent architecture.

- **Component-Based.** In component-based autonomic frameworks, autonomicity is achieved using modules that are not part of the system being monitored but can be *"plugged in"* to the system when needed. This encourages reusability and flexibility [29].

- **Technique-Based.** Technique-Based approaches utilize techniques from other areas of computing to achieve some level of autonomicity, e.g. Machine Learning, Data Mining, Artificial Intelligence (AI) and Control theory [29].

- **Service-Oriented.** These are autonomic systems designed to facilitate autonomicity in service-oriented environments, e.g. web services [29].

- **Injection of autonomicity.** In these systems an *"overlay"* is added to legacy systems which do not exhibit any autonomous behavior to achieve autonomicity. This is done without any change to the legacy system. These frameworks utilize techniques such as Case-Based Reasoning (CBR) to achieve autonomicity [29].

Based upon the hierarchy proposed by IBM, frameworks can also be grouped based on their level of autonomicity [22]. The five levels of autonomicity are described below.

- **Basic-Level 1.** At this level, all management tasks are carried out manually by system administrators. Some monitoring tools may be used but the information is not collected or analyzed using any intelligence. Most systems today are at this level of autonomicity [22].

- **Managed-Level 2.** At this level, the monitoring tools possess some degree of intelligence. They are able to analyze the information using some intelligence, reducing the amount of analysis the administrator needs to perform [22].

- **Predictive-Level 3.** At this level, the monitoring tools go beyond the primary analysis of system information as done with *"Managed"* systems. In addition, patterns of system behavior can be recognized and preemptive or remedial actions can be suggested by the system based on the previous action of administrators [22].

- **Adaptive-Level 4.** At this level, the system is better able to carry out some of the preemptive and remedial actions which can be suggested by a *Predictive* system [22].

- **Autonomic-Level 5.** At this level, full autonomicity is achieved. Systems can manage themselves dynamically using system information and business rules without the intervention of system administrators [22].

While a lot of work has been proposed in the area of self-healing, most of them do not place their work within the context of autonomic computing. A survey of the state of the art in self-healing in the context of autonomic computing shows that it still remains a long-term goal that is worth aiming for but which may not be achievable in the short-term [37]. For this reason this work focuses on only the detection aspects of self-healing, as these are achievable with current technology. So far most approaches to self-healing which place themselves within the context of autonomic computing utilize either Case-Based Reasoning (CBR), Artificial Intelligence (AI) or Machine Learning techniques [29] and most do not focus exclusively on the mining of application logs [22, 29]. The use of system metrics has received considerable attention as well [8].

The earliest work to recognize the importance of event messages for the goal of autonomic computing can be found in [71]. In that work, the author proposes a three-tiered data-driven approach to knowledge acquisition in autonomic systems. Rather than relying completely on human-generated rules for the autonomous system or computer-generated rules, the work proposes a compromise between the two extremes, i.e. human-assisted computer discovery and computer-assisted human discovery. This thesis continues along these lines by proposing a framework which can analyze a stream of messages- in this case the event log- and provide useful information to human operators for carrying out their network management function while relying as well on feedback from the operators to improve its detection capability.

A more detailed survey of previous work in automatic log analysis can be found in Chapter 2.

1.3 Motivation and Objectives

The overall goal of the research carried out in this thesis would be to design and evaluate a framework which utilizes a series of techniques to analyze the contents of system event logs automatically while focusing primarily on the discovery of patterns which are related to fault conditions, i.e. alerts. The analysis will take various forms but will utilize a mix of unsupervised and supervised techniques while keeping human involvement to a minimum. The components will be self-contained but could be interrelated as well, i.e. the output from one could serve as input to the other. The framework would enable system administrators to maximize event logs as a source of information in decision making, especially when faults occur on the system.

As this thesis is not the first work to attempt to do this, my goal is to go beyond the current state of the art by providing a framework which is not only more accurate and efficient but also requires minimal human input. The proposed system will provide a means for the automatic troubleshooting of fault situations in a network using the information contained in logs, thus contributing to autonomicity. With such a tool, the system could monitor its own logs automatically for symptoms of fault conditions. If these symptoms can be detected before actual faults occur, then proper remedial action can be taken. However if they are detected after the fault conditions occur, then they can help in reducing troubleshooting time. Either way, this will lead to significant cost savings for any enterprise.

In order to reach the main goal of this work, the following research objectives were set.

Minimum Apriori Information

Platform portability is important for any effective framework for alert detection in system logs. For a system to be platform portable, it must not rely on system specific characteristics and must assume very little knowledge of the architecture of the system it would be monitoring, i.e. minimum apriori information. Some previous approaches rely heavily on the knowledge of the systems that they will be monitoring, this reduces their platform portability. For this reason, this work will strive to produce a framework that requires minimum apriori information for operation. Furthermore, the system would also be developed to detect alerts without the use of semantic analysis. Analyzing terms semantically can lead to platform specific systems in this domain, therefore a system which utilizes semantic analysis in a significant way cannot be platform portable.

Unstructured Data

Log files contain unstructured data in the form of the natural language descriptions of events, i.e. messages. This unstructured content can serve as a stumbling block to automatic analysis. As stated in [21], abstracting these natural language descriptions is fundamental and contributes greatly to accuracy of analysis. However, abstraction is not a straightforward process; previous efforts show that there are several different approaches using such tools as Simple Logfile Clustering Tool (SLCT) and Loghound [78] which have been adopted for dealing with this problem.

This research will focus on the problem of the abstraction of the unstructured content of system logs. Specifically, the goal will be to develop techniques for extracting patterns from the logs which can be used in abstraction while ensuring that the extraction is done accurately and can be completed in a timely manner.

Interactive Learning

Due to the difficulty and uncertainty associated with automatic analysis performed by frameworks such as proposed by this thesis, the work in [3] recommends that human input be utilized by such systems. Such input could be achieved by way of a visualization system which allows human administrators to view the results of the analysis of the system and provide feedback which the system can use to carry out actions and improve on the analysis over time, i.e. interactive data mining via visualization. In previous research efforts where visualization has been used, the visualizations have only been used to view and analyze the system's output but not for providing feedback.

This thesis would therefore aim to develop a framework that includes an interactive feature as one of its components. This goal will coincide with the recommendation in [71], i.e. human-assisted computer discovery and computer-assisted human discovery.

Hybrid Detection

Systems which carry out automatic alert detection in system logs can be classified typically as either signature-based or anomaly-based detectors. Signature-based systems work by using well defined patterns of known previous alert activity for detecting alerts. Signature-based systems are generally very accurate but suffer from their inability to detect novel and unknown alert activity. Anomaly-based systems work by using statistical techniques to determine what is *normal*. In doing so, they can detect alerts by pointing out any activity which differs from the *norm*. Anomaly-based systems are generally not as accurate as signature-based systems: it is common for them to generate a lot of FPs. However, they are able to detect new and novel alert activity.

An ideal log alert detection framework should be able to detect new and novel alert activity while still achieving high accuracy in detection. To this end, one of the goals of the research carried out in this thesis is the designing and developing of a framework which builds on the strengths of both approaches. The resulting system would be a *hybrid* alert detection framework.

Self-Awareness

The four objectives of autonomic computing are self-configuration, self-optimization, self-healing and self-protection. Nonetheless, as noted in [13], there are four attributes which a true autonomic computing system must possess before any of the objectives can be achieved: self-monitoring, self-awareness, self-situation and self-adjustment. Most previous efforts have focused on achieving one or a combination of the objectives without focusing on the attributes. My aim is that this thesis will be able to contribute to self-awareness in computing systems by developing techniques which can be used to detect not just alert states but all system states which are discernible in a system log.

Computational Cost

One of the major hindrances to the automation of system logs is their size, which makes the application of certain techniques impracticable in this domain. Hence it is one of the goals of this thesis to ensure that most of the techniques used are simple and have low computational complexity in order to ensure that they scale gracefully when applied to large datasets. Most log files today can be expected to be large.

With these objectives in mind, the framework which this thesis aims to design and develop can be described as a hybrid interactive learning framework for alert detection in system logs. An overview of what the architecture of such a system may be is given in Fig. 1.3. At the core of the proposed system is its anomaly detection mechanism. The anomaly detection mechanism analyzes the contents of event logs, in the process, generating event log clusters which define the different system states or behaviors that are discernible in the log. It also determines whether these clusters are likely to contain normal or anomalous states. Then it presents these clusters to an administrator using a visualization system after which the administrator can then confirm the anomalous clusters detected by the system and provide labels. Subsequently, the system can then send the events in these labelled anomalous clusters to a signature generation system which generates a detection signature for the alert state(s) represented in the clusters. These signatures will be used to detect future occurrences of the alert state. Meanwhile, the anomaly detection component continues to use the feedback it gets from the administrator and the signature generation system to improve its anomaly detection capability.

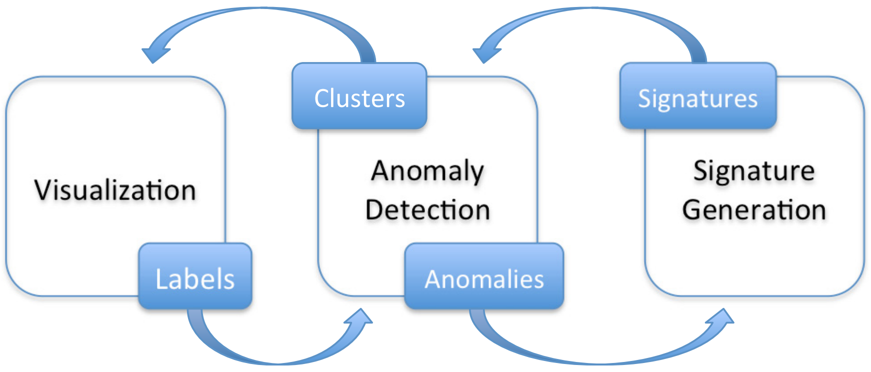


Figure 1.3: Overview of the hybrid interactive learning framework.

Based on the categorization types listed in Section 1.2, Table 1.1 places this proposal within the context of the current state of the art in autonomic computing. This table shows that the proposed system in this thesis fits very well within the context of current research in autonomic systems. Event logs contain information about the internal state and changes occurring in a system. By producing a framework which allows computer systems to monitor and analyze their event logs automatically for information about their current state, this thesis contributes to *self-awareness* and *self-monitoring* in autonomic systems. Again, by using this information to detect symptoms of failure automatically when they occur and inform the administrator accordingly, this thesis contributes towards the long-term goal of producing *self-healing* systems. Though security management is not addressed, this research could prove relevant in that domain. Hence, a secondary contribution of this research is *self-protection.*

The design approach is *external* as the framework performs all of its analysis outside of the system being monitored. By utilizing visualization, data mining and information-theoretic techniques, the framework can be described as *technique-based.* Most computing systems do not possess any degree of autonomicity, so this research aims to inject autonomicity into a non-autonomous system. A realistic level of autonomicity for this thesis is set as Level-3, which is achievable using today's technology. Autonomic computing research has been besieged with unrealistic expectations [13]. This implies that a more realistic and long-term approach has to be adopted to achieve the goal of creating systems at Level-4 and Level-5. Consequently, this proposal is therefore restricted accordingly.

Table 1.1: Placing the framework within the context of autonomic computing

Property	**Category**
Objective	Self-Healing, Self-Protection
Attribute	Self-Awareness, Self-Monitoring
Design Approach	External
Framework Type	Technique-Based, Injection of Autonomicity into Non-Autonomous Systems
Autonomicity Level	3-Predictive

1.4 Literary Contributions

A comprehensive list of the peer-reviewed literary contributions of this thesis can be found in Appendix E. The list includes two fully reviewed workshop papers, eight fully reviewed conference papers and one journal publication. Some of these publications are cited throughout the thesis at points where they are relevant.

1.5 Organization of the Thesis

The automatic log analysis framework proposed by this thesis deals primarily with alert detection. Therefore in the next chapter, Chapter 2, *alerts* as formulated by this research are defined. Previous and related work in automatic alert detection are discussed, while noting their limitations. The chapter goes on to introduce the anomaly detection component, which is at the core of the proposed alert detection framework, STAD (Spatio-Temporal Alert Detection).

Message type extraction is a fundamental task necessary for the meaningful automatic analysis of unstructured log data. Therefore, Chapter 3 of this thesis is devoted to automatic message type extraction. In this chapter, a novel linear-time algorithm for message type extraction developed as part of this thesis is introduced. IPLoM (Iterative Partitioning Log Mining), as the algorithm is called, is tested against previous approaches and is shown to outperform them.

In Chapter 4, practical applications of message types in event log management are presented. First, it is shown how message types can be used to enforce a hierarchy on the content of system logs. This hierarchy allows the contents of the log to be visualized interactively. A prototype log visualizing tool called LogView, which visualizes the hierarchy using treemaps [65] is presented as well. The scheme of this tool forms the visualization component of the alert detection framework. Second, a method for space efficient storage and easy retrieval of log events is demonstrated. The method is based on MTT. Storage of event logs using the format suggested here would ease some of the preprocessing required for automatic analysis. Third, the importance of message types to alert detection is demonstrated. Using the entropy-based alert detection mechanism of Nodeinfo [55], it is demonstrated that incorporating message types during model building not only results in more accurate models but can speed up the computation by up to a hundredfold.

Chapter 5 introduces the entropy-based ICC technique used to cluster log partitions as part of the STAD anomaly detection process. At the core of ICC technique is the theory that decomposing the contents of a system log spatio-temporally leads naturally to the grouping of correlated messages. This theory is developed in the chapter and is an important contribution of the thesis. This previously unknown property of system logs allows pseudo-linear discovery of correlated events, a process that typically requires significant computation.

Chapter 6 discusses STAD in more detail. A detailed evaluation involving log data for four HPC machines is presented. The data from these HPC machines represents ~81GBs of data collected from ~76,000 nodes over a cumulative period of ~3.5 years and ~746 million processor hours. This dataset, which is perhaps the largest dataset of its type publicly available at the time of writing this thesis, is appropriate for this research since it contains alerts which have been labelled using expert knowledge [56]. The chapter also explores the possibility of using automatic techniques such as C5.0 [62], for developing the rule base which is used by STAD to identify anomalous clusters.

As the focus of this thesis is not on alert detection in HPCs alone, Chapter 7 presents work evaluating STAD on non-HPC log data. The datasets were collected from a distributed system and a cloud-based system. This is appropriate as today's enterprise networks consist of both distributed and cloud-based systems. Specifically, the adoption of cloud computing infrastructures causes system logs to be multi-tiered. Thus, manual analysis of these logs proves to be difficult. System administrators of cloud computing systems are desperately in need of tools to make sifting through large volumes of inter-related log data an easier task. A data mining-based tool such as STAD would prove useful in the correlation of events across multiple tiers making log analysis efforts easier.

A detailed description and evaluation of the hybrid alert detection framework proposed by the thesis is presented in Chapter 8. The chapter shows how all the techniques introduced in previous chapters fit into the overall framework. The evaluations involved generating alert signatures automatically and testing them in a simulated online environment. The HPC, distributed and cloud logs were used for this purpose.

Finally, conclusions are drawn and future research directions are discussed in Chapter 9.

Chapter 2

Alert Detection in System Logs

Not all events in an event log are symptomatic of failure. Events which are symptomatic of failure or require the attention of administrators can be described as *alerts*. The task of *alert detection* can be defined as the task of identifying actionable events in an event log or identifying portions of an event log in which these actionable events are likely to exist [55]. The automation of *alert detection* is important, not only because it contributes to system dependability but because it contributes to the larger goal of building autonomic computer systems.

This thesis argues that there are two major types into which alerts in system logs can be categorized. These types, which will be referred to as Type-I and Type-II are described below:

- **Type-I:** These are alerts which are purely presence-based, i.e. these alerts can be identified simply by the presence of a specific signature (line or group of lines) in the log.

- **Type-II:** These are context-sensitive alerts, i.e. these alerts can be identified only by the context in which the events appear. Context-Sensitive alerts can be dependent on rates (i.e. the rate at which messages are generated), positional (i.e. where in the log the message is produced), or status based (i.e. the message identifies an alert only based on the operational status of its source) [55].

These alert types are not always mutually exclusive: an alert type can exhibit characteristics of both types. Any framework for alert detection needs to keep this distinction in mind.

This chapter of the thesis discusses previous/related work in automatic alert detection. It discusses the limitations of these approaches and then gives a quick overview of the anomaly detection mechanism which is at the core of the alert detection framework proposed by this thesis. The proposed anomaly detection mechanism is primed to discover mainly Type-I alerts but it is capable of detecting Type-II alerts as well. However, it may have difficulty detecting Type-II alerts which are purely rate sensitive.

2.1 Related and Previous Methods

In order to create applications which will be capable of monitoring systems automatically to identify alerts, access to a continuous stream of real-time data from the systems is mandatory. Such data must contain information explicitly or implicitly about the state of the system and its components.

A literature review shows significant effort has been carried out by various researchers in automatic alert detection and system monitoring and most have followed the approach of analyzing data from the system. Two important sources of data include system metrics [8, 53] and system logs[34, 33, 57, 78, 36, 55, 2, 15, 82, 21]. While these sources seem to be more prevalent in the literature, other sources have been explored. For example, Chen et al. model system behavior by collecting and modeling statistics on the paths which system requests follow as they move through a system [6]. They demonstrate how this knowledge can be used in failure and change management.

Metrics of system activity can be used in the automatic analysis of such systems. These metrics could be either low level system metrics or application level metrics. System metrics include measurements such as resource utilization (memory and CPU), length of system queues and latency of disk I/O operations, while application level metrics include transaction response time and request throughput [8].

Cohen et al., demonstrate a method of using system metrics to define system states [8]. Such observed states are clustered and indexed for similarity-based retrieval. Their assumption was that if an indexed system state can be used to identify a prior system problem, i.e. the state signifies an alert condition, then future system states, which are sufficiently *similar*, can be associated with the problem as well. In turn, this can lead to quick diagnosis and repair. Jiang et al. propose the utilization of entropy-based analysis of management metrics for the problem of fault detection [53]. Their work suggests that systems can be monitored by observing the changes in the *in-cluster* entropy of clusters identified using the normalized mutual information of metric pairs. In [25], the authors apply a dependency-aware framework based on the Tanimoto coefficient to the problem of fault diagnosis in enterprise software systems, using metric-correlation models built from collected application level metrics such as resource utilization and response time.

However, system logs seem to have received the most attention as a source of information for automatic alert detection and system monitoring. Hence, it is not surprising that there are several commercial and open source tools which aid system administrators in the monitoring of their event logs. Examples of such tools include Splunk [67] and Sisyphus [70].

Splunk is a proprietary indexing, search and analysis tool for unstructured data. Though used in the management of log data, it is designed for use on any form of unstructured data. The only task which is carried out automatically by Splunk is indexing. It does this by searching for fields in the data which it identifies through name/value pairs in the data. Also, It searches for well known fields which are associated with log data, e.g. IP addresses and hexadecimal digits. Any other field which it indexes must be specified by the user. Its search and analysis functionality are harnessed and controlled through the use of its search/query language and charting functionality. Its analysis functionality can be enhanced with the use of custom built tools. For alert detection it can identify alerts only as defined by users using regular expressions.

Sisyphus is a log data mining toolkit designed specifically with HPC logs in mind. However, its use is not limited to HPC logs alone. The system can be adapted for use on almost any type of log. This is done by providing it with custom parsers and adjusting its parameters. Just like Splunk it indexes the contents of log files to allow for faster searches of log contents and provides a charting/graphing capability as well. However, it goes a step further than Splunk by using information theory to rank the contents of logs for alert detection. Nodeinfo is one of the information theoretic algorithms which it utilizes. It uses the output of these information theoretic algorithms to color code terms in textual views of the logs which helps administrators to prioritize the contents of the log while viewing them. In addition, it automates the production of log parsing templates through the use of the Loghound log data mining tool [77]. However, the production of the templates is carried out only when an administrator requests it.

Nevertheless, these tools are still incapable of of automating the process of log monitoring and alert detection fully. Therefore, for the most part, these tasks are still carried out manually by system administrators. A lot of research is still required to develop tools and techniques which will bring the level of automation to the required level. These research efforts will have to focus on one or more of the problems listed below.

- **Unstructured Message Analysis.** Events in system logs are typically not homogenous entities. They contain structured and unstructured information. Typically, the unstructured information (namely free-form messages) pose a stumbling block to the automatic analysis of event data. Therefore, analysis of unstructured messages is required for further understanding of system logs. The methods proposed in [78, 45, 83, 2] show different approaches to carrying out unstructured message analysis.

- **Indexing/Feature Creation.** This involves the generation of indexable features from the unstructured data in the form of message type *IDs* and message variables. This is referred to as message type transformation (MTT) in this thesis.

- **Event Correlation.** In most cases, it is unlikely that a single event in a system log could characterize system behavior. As such, it is important to find message types which are correlated in the system logs. Correlated messages are usually better indicators of system state. In [83], the authors track the variables reported in message types as a means of identifying correlated messages. They argue that messages which report the same variable(s) are likely to be correlated.

- **Anomaly Detection.** If system states can be identified, the task of alert detection reduces to the task of identifying anomalous states. Thus, appropriate anomaly detection techniques need to be devised.

The research performed in this thesis intends to contribute to approaches for dealing with each of these problems. However, the following summarizes some previous work done with log data in system monitoring.

Loghound is a log data mining tool, which is an implementation of a frequent itemset mining algorithm for mining both lines (message types) and event types (correlated messages) from an event log [77]. In [78], a case study highlighting the use of Loghound to analyze Cisco Netflow logs for patterns which describe the behavior of network traffic is presented. In [34], Liang et al. propose a 3-step filtering algorithm for filtering failure logs from a high performance cluster (a BlueGene/L prototype), which compresses and categorizes the events in the log to understand failure behavior better.

In [33], the authors propose a modified Naive Bayes algorithm for the categorization of messages in system logs. Unlike other approaches in which such message categories are based on the textual representation of the messages, the categorization done here is based on previously defined categories associated with the IBM CBE (Common Base Event) format [74]. The authors then discover temporal relationships between these message categories. These relationships are then visualized to monitor system behavior.

In [57], the authors assume the existence of known *event types* (message types) and use the process they refer to as *event summarization* to mine and rank temporal dependencies between event types. Temporal dependencies are mined using time series analysis and are ranked using a *forward entropy* technique [9]. These dependencies are visualized using an Event Relationship Network (ERN) [72, 58], which is used to interpret system behavior and derive rules for system management. In [36], Lim et al. utilize message de-parameterization to create message types from enterprise telephony system logs. Then the messages in the logs are replaced by these message types, which they refer to as *message codes*. The logs are further analyzed using frequent itemset mining to discover correlated messages, which were useful in determining failure states in the system.

In [2] the authors propose a sequential algorithm for the discovery of message types. Being a sequential algorithm, it is suited for online discovery of message types. The authors also propose a novel algorithm; Principle Atom Recognition in Sets (PARIS), which uses the message types discovered to identify message types which tend to occur together in the event stream. Then these message types and correlations are used in the diagnosis of system problems and in the visualization of the log data.

In [83] the authors propose a framework for the detection of system problems through the mining of console logs. Using message types extracted directly from program source code, relevant features were extracted from event logs and processed using Principal Component Analysis (PCA). The PCA analysis was able to identify outliers (anomalies), which they showed corresponded with periods during which faults were injected into the network. This framework differs largely from the proposal in this thesis in many ways, most notably in the fact that the proposed framework extracts its message types directly from the event logs themselves, which is a more readily available extraction source. Also, in this thesis, the evaluation is carried out using data containing real systems faults, as opposed to injected faults.

In [21], the authors investigate the effects of message type transformation on the problem regarding the supervised learning of alert signatures from log files. The authors utilize and compare three associative classification methods, Naive Bayes classification and decision tree classification using C4.5 as supervised learning approaches. The associative classification methods used are, CBA (Classification-Based Association), CMAR (Classification based on Multiple Association Rules), and CPAR (Classification based on Predictive Association Rules).

They used data generated from a test bed having a web server running two applications and a database server, with faults simulated through fault injection. Their results suggest that apart from dimensionality reduction, message type transformation also helps to improve the accuracy of alert detection. Also, they found that associative classifiers worked better on logs which are not abstracted, while Naive Bayes and C4.5 performed better on abstracted logs. In contrast, in [15], the authors propose the use of Finite State Auotmata (FSA) as means for alert detection in logs.

On the other hand, Nodeinfo is an alert detection method which utilizes the entropy-based information content of system logs [69, 55]. Based on the assumption that *"Similar computers correctly executing similar code should produce similar logs,"* Nodeinfo introduces the more complex log.entropy term weighting scheme to the work of Liao [35] and Reuning [60]. Nodeinfo has been shown to achieve an operationally acceptable false positive rate of 0.05% at a Recall rate of 50% in the detection of alerts in system logs from HPCs. This thesis also builds on the entropy-based information content approach to system monitoring. In this thesis, I design and develop an entropy-based method which not only reduces the computational effort but also increases its detection accuracy. Moreover a temporal entropy element is introduced to the framework to deal with situations in which spatial-entropy alone is not sufficient for alert detection. Overall, the combination of these techniques produces a totally novel approach to alert detection which, while based on entropy-based techniques, is completely different in its own right.

2.2 Limitations of Current Methods

This section highlights some of the limitations of the current tools used in industry and previous research efforts. The system proposed in this thesis contributes to the literature by mitigating some of these limitations.

- **Limited automation.** While fully automated system monitoring and alert detection may not be possible yet, the level of automation found in most commercial tools still leaves much to be desired. Even when some automation is provided, the systems still require administrators to manage the automated process: e.g. automation of alert detection using signatures of known alerts require that administrators discover these signatures, input the signatures into a database and then manage the signature database over time. The proposed system in this thesis aims to improve the automation and to free administrators from some of the more mundane tasks involved in log monitoring.

- **Non-Interactive.** According to [71], an ideal system log monitoring tool should encourage co-operation between the autonomic system and human administrators. Most systems provide only part of this capability or do not provide it all. The use of visualization is restricted to aiding human administrators to comprehend the contents of the logs or for presenting the result of automatic analysis. The system receives no feedback which allows it to improve on its capabilities.

- **Signature or Anomaly Detection.** Most of the previous systems are either signature-based or anomaly-based. Signature-based systems tend to be very accurate but suffer from an inability to detect novel alert behaviour. On the other hand, anomaly-based systems are able to detect novel alert behaviour but are not as accurate as signature-based systems. An ideal system should take advantage of the strengths of both approaches. Consequently, a hybrid approach is proposed by this thesis.

- **Assumes knowledge of infrastructure.** Some previous approaches assume in-depth knowledge of the systems being monitored. This assumption limits the platform portability of such tools or techniques. While some knowledge of the monitored system is necessary, the proposed approach in this thesis attempts to limit the amount of information required to the barest minimum i.e. minimum apriori information. Minimum apriori information produces a system which assumes very little knowledge of the architecture of the system it would be monitoring and does not rely on system specific characteristics, producing a system which is platform portable.

2.3 The Proposed Anomaly Detection Framework

As discussed in Chapter 1, at the core of the alert detection framework is the anomaly detection mechanism. The novel framework proposed by this thesis for anomaly detection in system logs is called STAD. It works on the assumption that *"System logs events which are produced by similar spatial sources or produced during periods of similar system activity are likely to be similar,"* which is an extension on the assumption used in entropy-based systems mentioned earlier in this chapter.

The main contributions of the proposed system are: (i) the extension of entropy-based approaches by allowing the detection of alerts without resorting to ranking schemes; and (ii) the enabling the analysis of a group of dissimilar nodes, which means that it can be applied to distributed systems. The best case results for STAD also improve on the baseline alert detection of an already deployed system, i.e. Nodeinfo, on the same datasets. Published best case results for Nodeinfo show 50% detection with a false positive rate of 0.05% [69]. My research is also the first to provide average case results across several datasets for an entropy-based approach in which the entire evaluation is carried out automatically. In previous works, evaluation results were determined manually from a Precision-Recall plot [55, 47].

An overview of the approach is provided in Fig. 2.1. Overall, each step of the approach is general enough to allow flexibility in the choice of methods used. STAD is referred to as an entropy-based approach in this thesis because some of the steps used in the implementation utilize entropy-based approaches. However, the STAD framework does not need to be entropy-based; it can be implemented without the use of entropy-based techniques. The phases in the proposed framework are described in detail in Sections 2.3.1 and 2.3.2. Moreover, the specific techniques employed for the various components of STAD are presented in the following chapters, while the framework itself is discussed in detail in Chapter 6.

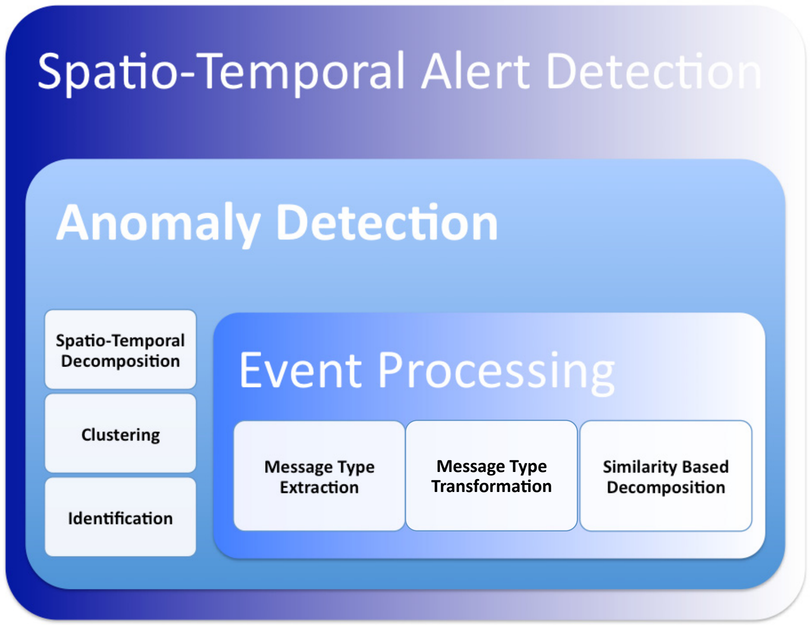

Spatio-Temporal Alert Detection
Anomaly Detection
Spatio-Temporal Decomposition
Clustering
Identification
Event Processing
Message Type Extraction
Message Type Transformation
Similarity Based Decomposition

Figure 2.1: Overview of the STAD framework.

2.3.1 Event Processing

As hinted in previous work [32, 84, 56], extracting sufficient structure from event log data can be beneficial to automatic analysis. Essentially, this step of the framework carries out the task of extracting structure from the unstructured component of the logs. Event logs contain semi-structured information from heterogenous sources. This fact coupled with the ambiguity introduced by the semantics of terms used by application developers makes this step important. The event processing phase of the framework is sub-divided into three components which are Message Type Extraction, Message Type Transformation and Similarity-based Decomposition.

1. **MessageType Extraction.** A single event line in a log file consists of different fields (see Fig. 1.2 in Chapter 1). The *"Message"* is perhaps the most unstructured field within an event. They are produced by *"print"* statements within program code. The goal of message type extraction is to extract structure from the message field of events. Since messages are produced by print statements within code, grouping messages produced by the same print statements is a good way to find such structure automatically.

2. **Message Type Transformation.**

   This component of the event processing phase is intended to apply the discovered structure to the events in the log file. Using the message type templates extracted from the message type extraction step, the messages in the event data are transformed so that a more concise representation is achieved. This concise representation has the effect of not only imposing structure on the unstructured messages in the event data, but also of contributing to data compression.

3. **Similarity-based Decomposition.** This step of the process attempts to decompose the log into more homogenous units to allow for useful analysis. This decomposition can be done using any means, depending on the goal of the analysis.

2.3.2 Anomaly Detection

The goal of this phase of the framework is to identify the portions of the log which contain anomalous events. This phase has three components which are Spatio-Temporal Decomposition, Clustering and Identification.

1. **Spatio-Temporal Decomposition.** This component decomposes the content of any log based on source and time information. After decomposition each resultant unit will contain log information from a single source over a unit of time. This thesis argues that this action leads to the discovery of correlated message types.

2. **Clustering.** The goal here is to group the spatio-temporal units such that members of each group are very similar to each other while being very dissimilar from the members of other groups. The resulting groups are referred to as clusters.

3. **Identification.** This step aims to identify those clusters containing spatio-temporal units which have anomalous activity. This step completes the process of anomaly detection.

Chapter 3

Message Type Extraction

A basic task in the automatic analysis of log files is message type extraction [30, 40, 86, 68]. Message types are groups of unstructured natural language event descriptions in event logs which have the same or similar semantic meanings. Message type descriptions on the other hand are textual templates which abstract all the instances of a message type. Unfortunately, the message types which exist in an event log are not always known apriori. Message type extraction is the task of finding message types when they are not known. The unstructured content of event logs constitute a key challenge to achieving fully automatic analysis of system logs. Message types, once found, can help to mitigate this problem. Apart from this, message types are useful for the following.

- **Compression.** Message types can abstract the contents of system logs. Therefore, they can be used to obtain more concise and compact representations of log entries. This leads to memory and space savings.

- **Indexing:** Each message type can be assigned a unique *ID (Identifier Index)*, which in turn can be used to index historical system logs leading to faster searches.

- **Model Building.** Building computational models using log data usually requires the input of structured data. This process can be facilitated by the initial extraction of message type information. Message types can be used to impose structure on the unstructured messages in the log data before they are used to build computational models. In [83, 15], the authors demonstrate how message types can be used to extract measured metrics used for building computational models from event logs. The authors were able to use their computed models to detect faults and execution anomalies using the contents of system logs.

- **Visualization.** Visualization is an important component of the analysis of large data sets. Visualization of the contents of systems logs can be made more meaningful to a human observer by using message types as a feature of the visualization. For the visualization to be meaningful to a human observer, the message types must be interpretable. This fact provides a strong incentive for the production of message types which have meaning to a human observer.

As this thesis deals with the automatic analysis of log files, it would be impossible to complete it without touching on the subject of message type extraction. This chapter highlights a novel algorithm for message type extraction: Iterative Partitioning Log Mining (IPLoM). It showcases the results of benchmarking it against some well known techniques for message type extraction.

The main assumptions on the kind of event logs that IPLoM is suitable for are listed below.

1. The events in the log contain at least one field which is an unstructured natural language description of the event. These descriptions, which are called "messages," illustrated in Fig. 3.1, would be produced naturally by a set of "print" statements from a source code.

2. The underlying structure of these "messages" is unknown or not well documented.

The following points highlight how IPLoM differs from previous methods for message type extraction:

- IPLoM does not assume any knowledge of the underlying system which will produce the event log (e.g source code).

- IPLoM's results tend to match message types produced by humans more closely than previous approaches.

- IPLoM is able to find both frequent and infrequent message types in the log data. This is important as rare patterns are often needed for certain types of analysis.

The material presented in this chapter can be found in these publications [45, 43].

3.1 Background and Motivation

An example event from an event log which uses the syslog format [38] is given in Fig. 3.1. A line in a event log is not a homogenous entity; it consists of several fields with different levels of structure. The natural language ”MESSAGE” field which consists of a variable number of tokens is the most unstructured portion of the log. It describes the incident which triggered the event. The unstructured nature of this field is a major impediment to the fully automatic analysis of event logs. Message type extraction is an attempt to find structure within the unstructured natural message descriptions in event logs.

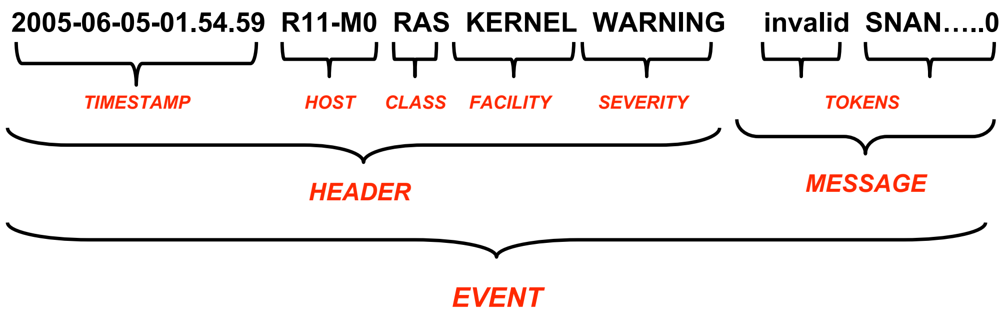


Figure 3.1: An example system log event.

To give an example of what message types are, consider this line of code:
`sprintf(message, Connection from %s port %d, ipaddress, portnumber);`
in a *C* program. This line of code could produce the following log messages:

```
  ‘‘Connection from 192.168.10.6 port 25’’
‘‘Connection from 192.168.10.6 port 80’’
‘‘Connection from 192.168.10.7 port 25’’
‘‘Connection from 192.168.10.8 port 21’’.
```

These four messages would form a message type in the event log and can be represented by the message type description:
`‘‘Connection from * port *’’ .`
The wildcards, i.e. the “*” characters, represent placeholders which can match any word and are referred to as message variables. This representation of message types

will be adopted in the rest of the thesis. Determining what constitutes a message type might not always be as simple as this example might suggest. Consider the following messages produced by the same print statement. `‘‘Link 1 is up’’`, `‘‘Link 1 is down’’`, `‘‘Link 3 is down’’`, `‘‘Link 4 is up’’`. The most logical message type description here is `‘‘Link * is *’’`, however, from an analysis standpoint, having two descriptions `‘‘Link * is up’’` and `‘‘Link * is down’’` may be preferable. There may be other cases as well in which messages produced by different print statements could form single logical message types. However, for the most part, message types will usually correspond to messages produced by the same print statement, so this definition is retained for simplicity.

This message type extraction problem is well attested to in the literature but there is as yet no standard approach to the problem [83]. Techniques for automatic message type extraction are varied. Some of these approaches will be described subsequently, but it is important to note that some event log formats are well structured and well documented (e.g. IBM Webshpere). In such cases, message type extraction may not be necessary. For example in [81], the authors provide an example of process mining in web services using Websphere, while in [11], the authors propose a tool for visualizing web services by mining the contents of event logs. Process mining refers to the analysis of event logs as a way of monitoring adherence to business process rules. The analysis can be carried out without message type extraction due to the structured nature of the Webshpere logs which utilize the CBE model[74] for event representation.

Approaches to message type extraction include extraction from source code [83] and the use of techniques designed for pattern discovery in other types of textual data, such as the use of Teiresias [61, 68]. However, the most popular approach has been to view the message extraction task as a data clustering problem while using frequent itemset mining approaches as a means of discovering the clusters.

If each textual line in an event log is considered a data point and its individual words considered attributes, then the job of message type extraction can be seen as the task of grouping similar log messages together. This is a clustering task which involves using the words in each line as attributes. For example the log message *Command has completed successfully* can be considered a data point with four dimensions, having the following attributes *"Command," "has," "completed," "successfully"*. However, as stated in [76], traditional clustering algorithms are not suitable for event logs for the following reasons.

1. The event lines, do not have a fixed number of attributes.

2. The data point attributes, i.e. the individual words or tokens on each line, are categorical. Most conventional clustering algorithms are designed for numerical attributes.

3. Event log lines are high dimensional. Traditional clustering algorithms are not well suited for high dimensional data.

4. Traditional clustering algorithms also tend to ignore the order of attributes. In event logs, the order of the attributes is important.

For these reasons various domain specific data clustering algorithms for event logs have been developed. Examples of such algorithms include Simple Logfile Clustering Tool (SLCT) [76] and Loghound [77]. SLCT and Loghound are similar to the Apriori algorithm [1].

In order to evaluate the performance of IPLoM, publicly available implementations of SLCT, Loghound and Teiresias were selected.

3.2 Methodology

In this section the message type extraction techniques against which IPLoM is benchmarked are described in more detail.

3.2.1 Simple Logfile Clustering Tool (SLCT) and Loghound

SLCT and Loghound both work by viewing the message type extraction problem as a frequent itemset mining problem. Given a set of items (called an item base), S, let us define $\mathbf{T}$ as a set of transactions defined over S such that for all $T \in \mathbf{T}$, $T \subseteq S$. A subset (also called an itemset) S' of S is said to be *frequent* if the number of transactions in $\mathbf{T}$ which are supersets of S' exceed a user-specified support threshold and is referred to as a *frequent itemset*. Hence, the frequent itemset mining problem can be defined as the task of finding all frequent itemsets which have support above a specified threshold value in a given transaction database, **T**.

The algorithms view each line in the event log as a transaction containing an ordered list of items, the items being the words in the log line. Since message types contain words which occur frequently together on each line, then finding the frequent itemsets of words would reduce to the problem of message type extraction.

Itemsets can be defined by the number of items they contain; e.g a *k-itemset* is one which contains k items. While searching for the frequent itemsets, SLCT and Loghound differ fundamentally in that while SLCT only considers 1-itemsets, Loghound considers all itemsets up to size n, n as defined by the user.

Frequent itemset mining can be a very memory intensive process due to the large number of items which may exist in the transaction database. A large number of items means that a large number of candidate itemsets would be generated during the mining process. The number of candidate itemsets increases exponentially with respect to the number of items in the database. This problem can be especially severe when dealing with log files. Log files may have unique words numbering in the millions. Thus, SLCT and Loghound utilize two properties of event logs to reduce the memory requirements of the search. These properties are:

1. **A majority of the words in an event log occur infrequently, sometimes no more than once and only a small fraction occur frequently.** This property allows the use of fixed size, m, a summary vector to estimate the words which do not need to be stored in memory. A string hashing function which returns a value between 0 and $m - 1$ is applied to each word in the log file. Each of the m counters of the summary vector keeps a count of the number of times a word hashes to that value. Eventually, when the actual vocabulary gets built only those words which hash to values that exceed the support value in the summary vector are utilized, thus reducing significantly the memory requirements of storing the list of items.

2. **A strong correlation exists among frequent words.** This is due to the fact that every line in an event log is formatted according to a format string. Therefore the constant words in the format string end up occurring frequently in log files. If the maximum number of words which occur on any line in the log

file is k, then all itemsets of size between 1 and k need to be generated to search for all the frequent itemsets. However, with this property, it is possible to search for itemsets of a size smaller than k and still produce appreciable results. This strong correlation which exists among frequent words, implies that combinations of small size frequent itemsets lead naturally to frequent itemsets of a larger size. This allows the search to be completed while generating only a few candidate itemsets.

SLCT's three step process will now be discussed in more detail.

1. The first step is the data summarization phase, which involves two passes over the data. In the first pass, the summary vector is built and the list of the potential 1-itemsets is selected for the vocabulary. The second pass identifies the frequent words in the vocabulary. These frequent words are frequent 1-itemsets. SLCT and Loghound make the distinction of differentiating words by their position on the line. For instance, if the word *"pass"* appears as the second and third words on two separate lines, then the following word and position pairs will be created to separate the two instances: i.e. *(2, "pass") and (3, "pass").*

2. In the next phase another pass is made over the data to determine the frequent itemset candidates. The set of candidates is initially empty; as the data is scanned, the 1-itemsets which occur on each line are determined. A candidate which is defined by the combination of 1-itemsets that occur on a line is formed. If the candidate already exists then its counter is incremented, otherwise it is inserted in the list of candidates and its counter set to 1.

3. In the final step, the list of frequent itemset candidates is inspected. All candidates which have a count value which exceeds the support value are selected. These selected candidates are the frequent line patterns (message types) which occur in the log. For example, a selected candidate defined by the 1-itemsets, {*(1, "DHCPACK"), (2, "for")*}, will correspond to the message type
‘‘DHCPACK for *” .

A not-so-obvious advantage of determining frequent itemsets through a combination of 1-itemsets, rather than through actual candidate generation, is that the algorithm detects mainly maximal frequent itemsets. For instance, for the message type ‘‘DHCPACK for *” , the subtypes ‘‘DHCPACK * *” and ‘‘* for *” are also potential message types. By printing out only the first pattern, the number of message types produced is reduced and the user is not overwhelmed with the size of the output.

An open source implementation of SLCT written in C can be downloaded from here [80]. A complete list of the options which can be used with the tool are listed in Section A.1.1 of Appendix A.

Loghound differs from SLCT both in its utility and in the way it determines frequent patterns in the log. Apart from finding the frequent line patterns which occur in the log, Loghound is able to find sets of correlated message types as well, i.e. message types which occur frequently in close temporal proximity in the log. A frequent itemset mining paradigm is used for this task as well.

Loghound discovers frequent line patterns in much the same way as SLCT. It follows the same 3-step process but it can utilize itemsets of a size larger than 1 for candidate line pattern generation. As discussed, itemset generation can be memory intensive. Therefore, to ensure that minimal memory is used, the implementation of Loghound borrows ideas from the FP-growth [18] and Eclat [85] algorithms. Loghound generates itemsets in a tree-like data structure called an itemset trie, which it explores in a breadth first search manner.

Just like SLCT, Loghound produces output that is a series of line patterns, Fig. 3.2 shows four examples of the type of clusters which SLCT and Loghound are able to find. With both SLCT and Loghound, lines which do not match any of the frequent patterns discovered are termed *“outliers”*.

```
Feb * * big sshd[*]: Invalid user * from *
* - - [* -0500] "GET *
Oct * * big pop3d: LOGOUT, ip=[*]
* * * big pop3d: Connection, ip=[*]
```

Figure 3.2: Sample clusters generated by SLCT and Loghound

The Loghound implementation is also open-source and written in C. It can be downloaded from the url a this reference [79]. A complete list of the options which can be used with the *"Loghound"* tool once it is complied is provided in Section A.1.2 of Appendix A. SLCT and Loghound have received considerable attention and have been used in the implementation of the Sisyphus Log Data Mining toolkit [70], in online failure prediction [64] and in intrusion detection [20].

3.2.2 Teiresias

Teiresias is a bio-informatics pattern discovery algorithm developed by IBM [61]. It discovers motifs in biological sequences (protein or gene sequences). A motif is a subsequence of characters which occur very frequently within the biological sequence. It differs from previous algorithms for motif discovery in biological sequences in that it is able to find all the patterns of interest without having to enumerate the entire solution space, a property which greatly enhances its performance. Another difference is that it is guaranteed to list only maximal patterns.

The algorithm works in two phases. During its first phase, called the scanning phase, its finds all elementary patterns which meet minimum support. In other terms, it takes a set of strings X and breaks them up into a set of unique characters C, which are the building blocks of the strings. Only characters which meet a minimum support value K are listed in C.

In the second phase, called the convolution phase, these elementary patterns are combined successively into larger patterns until all maximal patterns have been generated. The maximal patterns must have at least a specificity determined by L/W, where L is the number of non-wildcard characters from C and W is the width of the motif with wildcards included.

Teiresias can be applied to message type extraction if message type templates are viewed as motifs which appear within a set of strings, i.e. the lines in the event log. A publicly accessible web-based implementation of Teiresias is available at this URL [23]. The most important parameters for running the algorithm from the web interface are L, W and K as described previously.

3.3 The Proposed technique: IPLoM Algorithm

Previous tools for message type extraction such as SLCT and Loghound have limitations. The development of IPLoM was carried out with the aim of producing a message type extraction algorithm which overcomes these limitations. These limitations are itemized below.

- They are unable to find all the possible message types which appear in a log data file. The techniques used by these tools focus on the frequency of the terms which appear in the data, hence can find only those message types which appear frequently. IPLoM is designed to find both frequent and infrequent message types.

- The support thresholds of SLCT and Loghound can be altered to allow them to find infrequent message types. However, tests have shown that even when this is done, the tools continue to show a bias for finding the frequent message types. IPLoM is designed not to have a bias for either frequent or infrequent message types. It gives all message types an equal chance of being found.

- A factor in judging the usability of the automatically extracted message types is readability, i.e. the ability of a human observer to view and make sense of them. Sometimes SLCT and Loghound produce message types at an abstraction level that makes it difficult for a human to make sense of them. Hence, a design goal for IPLoM is to produce message types at an abstraction level preferred by a human observer.

The IPLoM message type extraction works by partitioning a set of log messages iteratively. At each step of the partitioning process, the resultant partitions come closer to containing only log messages which belong to the same message type. By partition, a non-overlapping group of messages is assumed. At the end of the partitioning process the algorithm attempts to discover the template which generalizes the log messages in each partition: these discovered partitions and templates are the output of the algorithm.

The algorithm works through a four step process. An outline of the four steps of IPLoM is given in Fig. 3.3. The algorithm is designed to discover all message types in the set of log messages and does not require a support threshold as do SLCT or Loghound. Since it may be necessary occasionally to find only message types which have a support that exceeds a certain threshold, a file prune function (Algorithm 1) is incorporated into the algorithm. By removing the partitions which contain a number of messages which fall below the threshold value at the end of each partitioning step, only message type descriptions which meet the desired support threshold are included in the output. However, the use of the file prune function is optional. The following sub-sections describe each step of the proposed algorithm in more detail.

Algorithm 1 File_Prune Function: Prunes the partitions produced using the file support threshold.

```
Input: Collection $C[]$ of log file partitions.
    Real number $FS$ as file support threshold. {Range for $FS$ is assumed to be between $0 - 1$.}
Output: Collection $C[]$ of log file partitions with support greater than $FS$.
1: for every $partition\ in\ C$ do
2:    $Supp = \frac{\#LinesInPartition}{\#LinesInCollection}$
3:    if $Supp < FS$ then
4:       Delete partition from $C[]$
5:    end if
6: end for
7: Return(C)
```

3.3.1 Step 1: Partition by event size.

The first step of the partitioning process works on the assumption that log messages which belong to the same message type are likely to have the same event size. The event in Fig. 3.1 has an event size of 2. For this reason IPLoM's first step (Fig. 3.4) uses event size as a heuristic to partition the log messages.

Consider the message type description *"Connection from *"*, which contains 3 tokens. It can be be concluded intuitively that all the instances of this message

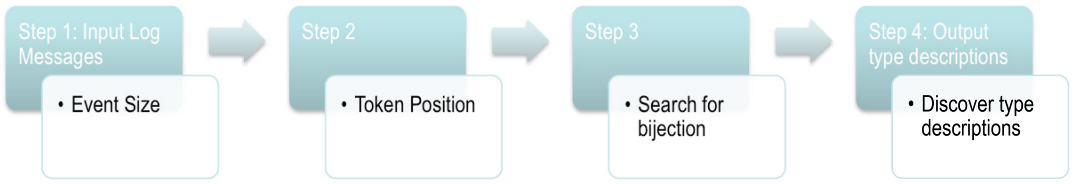

Step 1: Input Log Messages
• Event Size
Step 2
• Token Position
Step 3
• Search for bijection
Step 4: Output type descriptions
• Discover type descriptions

Figure 3.3: Overview of IPLoM processing steps.

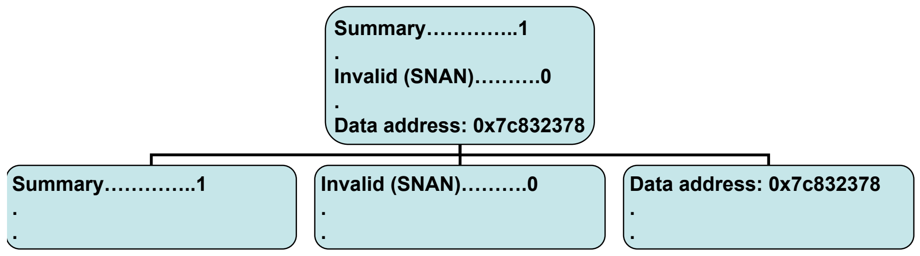


Figure 3.4: IPLoM Step-1: Partition by event size. Separates the messages into partitions based on the their event sizes.

type e.g. *"Connection from 255.255.255.255"* and *"Connection from 0.0.0.0"* would contain the same number of tokens as well. By partitioning the data first by event size, the property of most message type instances having the same event size is being taken advantage of. Therefore, the resultant partitions of this heuristic are likely to contain the instances of the different message types which have the same event size. A detailed description of this step of the algorithm is given in Algorithm 2.

Additional heuristics are used in the remaining steps to divide the initial partitions further. The partitioning process induces a hierarchy of maximum depth 4 on the messages and the number of nodes on each level is data dependent.

Sometimes, it is possible that message types which contain instances of variable size exist in the event log. This scenario is explained in more detail in Section 3.5.3. Since IPLoM assumes that messages belonging to the same message type should have the same number of tokens or event size, this step of the algorithm would separate instances of such message types. This does not occur very often and variable size message types can be found by post processing IPLoM's results. The process of finding variable size message types can be computationally expensive. Nevertheless, performing this process on the templates produced by IPLoM rather than on the complete log would require less computation.

Algorithm 2 IPLoM Step 1: Partition by Event Size

```
Input: Log file containing log messages.
Output: Collection $C$ of log file partitions.
1: for each line in the log file do
2:     Determine the count of tokens in line as $token\_count$. {Token delimiter is assumed
       to be space character.}
3:     if partition for lines with $token\_count$ tokens exists then
4:         Add line to the appropriate partition.
5:     else
6:         Create partition for lines with $token\_count$ tokens.
7:         Add line to the appropriate partition.
8:     end if
9: end for
10: Add all partitions to $C$.
11: $C = File\_Prune(C)$
12: Return(C)
```

3.3.2 Step 2: Partition by token position.

At this point, each partition of the log data contains log messages which are of the same size and therefore, the log messages can be viewed as n-tuples, with n being the event size of the log messages in the partition. This step of the algorithm works on the assumption that the column with the least number of variables (unique words) is likely to contain words which are constant in that position of the message type description which produced them. Therefore the heuristic is to find the token position with the least number of unique values and split each partition using the unique values in this token position, i.e. each resulting partition will contain only one of those unique values in the token position discovered, as can be seen in the example outlined in Fig. 3.5. A pseudo-code description of this step of the partitioning process is given in Algorithm 3.

The memory requirement of unique token counting is a potential concern with this step of the algorithm. While the problem of unique token counting is not specific to IPLoM, however, IPLoM has an advantage in this respect. Since IPLoM partitions the data, only the contents of the partition being handled need to be stored in memory.

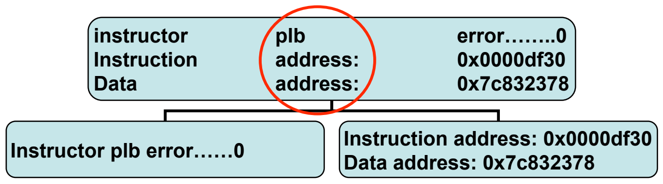

instructor plb error.......0
Instruction address: 0x0000df30
Data address: 0x7c832378
Instructor plb error......0
Instruction address: 0x0000df30
Data address: 0x7c832378

Figure 3.5: IPLoM Step-2: Partition by token position. Selects the token position with the least number of unique values, token position 2 in this example. Then, it separates the messages into partitions based on unique token values, i.e. "plb" and "address:", in the token position.

This reduces the memory requirements of the algorithm greatly. Moreover, other workarounds can be implemented to reduce the memory requirements further. For example, in Step-2 of the algorithm, by determining an upper bound on the lowest token count in Step-1, the memory requirements of this step can be significantly reduced. Further counts of unique tokens in any token position which exceeds the upper bound can be eliminated. However, in this research, the aim is to make a proof of concept so we left the implementation of such code optimization techniques for future work.

Despite the fact that the token position with the least number of unique tokens is used, it is still possible that some of the values in the token position might be variables in the actual message type descriptions. While an error of this type may have little effect on Recall, which measures the ratio of relevant items retrieved to the entire set of relevant items in a retrieval task, it could affect Precision adversely, which measures the ratio of relevant items retrieved in a set of retrieved items for a retrieval task. To mitigate the effects of errors of this nature, a partition support ratio (PSR) for each partition produced could be introduced. The PSR is calculated as in Eq. 3.1. Then a partition support ratio threshold (PST) can be defined . We group any partition with a PSR which falls below the PST into one partition (Algorithm 3). The intuition here is that a child partition which is produced using a variable token value may not have enough lines to exceed a certain percentage (the partition support ratio threshold) of the log messages in the parent partition. It should be noted that this threshold is not necessary for the algorithm to function and is introduced only to give the system administrators the flexibility to influence the partitioning based

Algorithm 3 IPLoM Step 2: Selects the token position with the lowest cardinality and then separates the lines in the partition based on the unique values in the token position. Backtracks on partitions with lines which fall below the partition support threshold.

```
Input: Collection of log file partitions from Step-1.
    Real number $PST$ as partition support threshold. {Range for $PST$ is assumed to be
    between $0 - 1$.}
Output: Collection of log file partitions derived at Step-2 $C\_In$.
 1: for every log file partition do {Assume lines in each partition have same event size.}
 2:   Determine token position $P$ with lowest cardinality with respect to set of unique
      tokens.
 3:   Create a partition for each token value in the set of unique tokens which appear in
      position $P$.
 4:   Separate contents of partition based on unique token values in token position $P$. into
      separate partitions.
 5: end for
 6: for each partition derived at Step-2 do {}
 7:   if $PSR < PS$ then
 8:     Add lines from partition to Outlier partition
 9:   end if
10: end for
11: File_Prune() {Input is the collection of newly created partitions}
12: Return() {Output is collection of pruned new partitions}
```

on expert knowledge they may have to avoid errors in the partitioning process.

$$PSR = \frac{\#LinesInChildPartition}{\#LinesInParentPartition} \tag{3.1}$$

3.3.3 Step 3: Partition by search for bijection

In the third and final partitioning step, partitioning is done by searching for bijective relationships between the set of unique tokens in two token positions selected using a heuristic as described in Algorithm 4. Consider the example messages below as a log partition.

```
Command has completed successfully
Command has been aborted
Command has been aborted
Command has been aborted
Command failed on starting
```

This partition has an event size of *4*. The token positions on which to perform the search for bijection need to be selected. The first token position has one unique token, {Command}. The second token position has two unique tokens, {has, failed}. The third token position has three unique tokens, {completed, been, on}. While the fourth token position has three unique tokens, {successfully, aborted, starting}. We notice in this example that token count *3* appears most frequently, i.e. twice: once in position *3* and once in position *4*. Therefore, the heuristic would select token positions *3* and *4* in this example.

To summarize the steps of the heuristic, first we determine the number of unique tokens in each token position of a partition. Then we determine the most frequently occurring token count among all the token positions. This value must be greater than 1. The token count which occurs most frequently is likely indicative of the number of message types which exist in the partition. If this is true, then a bijective relationship should exist between the tokens in the token positions which have this token count. Once the most frequently occurring token count value is determined, the token positions chosen will be the first two token positions which have a token count value equivalent to the most frequently occurring token count.

A bijective function is a *1-1* relation which is both injective and surjective. When a bijection exists between two elements in the sets of tokens, usually it implies that a strong relationship exists between them and log messages which have these token values in the corresponding token positions. Thus, log messages which have these token values in the chosen token positions are separated into a new partition.

Sometimes the relations found are not *1-1* but *1-M*, *M-1* and *M-M*. In the example given in Fig. 3.7, the tokens *privileged* and *instruction* with the tokens *imprecise* and *exception* have a 1-1 relationship because all lines which contain the tokens *privileged* and *imprecise* in position 2 also contain the tokens *instruction* and *exception* in position 3 as well and vice versa. Consider the event messages given in

Fig. 3.3.3 below to illustrate 1-M, M-1 and M-M relationships. If token positions 2 and 3 are chosen by the heuristic, a 1-M relationship with tokens *speeds*, *3552* and *3311* will be found, as all lines which contain the token *speeds* in position 2 have either tokens *3552* or *3311* in position 3, a M-1 relationship will be the reverse of this scenario. On the other hand, if token positions 3 and 4 are chosen by the heuristic, a M-M relationship will be found.

```
Fan speeds 3552 3552 3391 4245 3515 3497
Fan speeds 3552 3534 3375 4787 3515 3479
Fan speeds 3552 3534 3375 6250 3515 3479
Fan speeds 3552 3534 3375 **** 3515 3479
Fan speeds 3311 3534 3375 4017 3515 3479
```

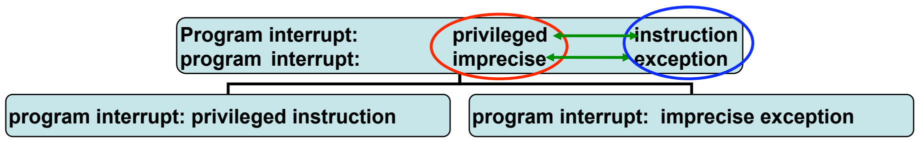
Figure 3.6: Example messages illustrating 1-M, M-1 and M-M relationships.

It is obvious that no discernible relationship can be found with the tokens in the chosen positions. Token *3552 (in position 3)* maps to tokens *3552 (in position 4)* and *3534*. On the other hand, token *3311* maps to token *3534* as well which makes it impossible to split these messages using their token relationships. Such a scenario is referred to as a M-M relationship.

In the case of *1-M* and *M-1* relations, the *M* side of the relation could represent variable values (so only one message type description is being dealt with) or constant values (so that in fact each value represents a different message type description). The diagram in Fig. 3.8 describes the simple heuristic which is used to deal with this problem. Using the ratio between the number of unique values in the set and the number of lines which have these values in the corresponding token position in the partition, and two threshold values, a decision is made on whether to treat the *M* side as consisting of constant values or variable values. *M-M* relationships are split iteratively into separate *1-M* relationships or ignored depending on whether the partition is coming from Step-1 or Step-2 of the partitioning process, respectively.

Before partitions are passed through the partitioning process of Step 3 of the algorithm, they are evaluated to determine whether they form good clusters already. To do this, a cluster goodness ratio threshold (CGT) is introduced into the algorithm. The cluster goodness ratio (CGR) is the ratio of the number of token positions which have only one unique value to the event size of the lines in the partition, according to Eq. 3.2. In the example in Fig. 3.7, the partition to be split has four token positions. Of these four, the first and second have only one unique value, i.e. “Program” and “Interrupt” respectively. Therefore, the CGR for this partition will be $\frac{2}{4}$. Partitions which have a value higher than the CGT are considered good clusters and are not partitioned any further in this step. Just as in Step-2, the PSR can be used to backtrack on the partitioning at the end of Step 3. While the backtracking is optional as with Step 2, it is advisable that it is carried out to deal with errors when they occur.

$$CGR = \frac{\#TokenPositionsWithOneUniqueTokenInPartition}{\#EventSizeOfPartition} \tag{3.2}$$



Program interrupt: privileged instruction
program interrupt: imprecise exception
program interrupt: privileged instruction
program interrupt: imprecise exception

Figure 3.7: IPLoM Step-3: Partition by search for bijection.

Algorithm 4 IPLoM Step 3: Selects the two token positions and then separates the lines in the partition based on the relational mappings of unique values in the token positions. Backtracks on partitions with lines which fall below the partition support threshold.

```
Input: Collection of partitions from Step 2. {Partitions of event size 1 or 2 are not processed here}
  Real number $CT$ as cluster goodness threshold. {Range for $CT$ is assumed to be between $0-1$.}
Output: Collection of partitions derived at Step-3.
1: for every log file $partition$ do
2:   if $CGR >= CT$ then {See Eq. 3.2}
3:     Add partition to collection of output partitions
4:     Move to next partition.
5:   end if
6:   Determine token positions using heuristic as $P1$ and $P2$. {Heuristic is explained in the text.
     Token position $P1$ is assumed to occur before $P2$.}
7:   Determine mappings of unique token values $P1$ in respect of token values in $P2$ and vice versa.
8:   if mapping is $1-1$ then
9:     Create partitions for event lines that meet each $1-1$ relationship.
10:  else if mapping is $1-M$ or $M-1$ then
11:    Determine variable state of $M$ side of relationship.
12:    if variable state of $M$ side is $CONSTANT$ then
13:      Create partitions for event lines that meet relationship.
14:    else {variable state of $M$ side is $VARIABLE$}
15:      Create new partitions for unique tokens in $M$ side of the relationship.
16:    end if
17:  else {mapping is $M-M$}
18:    All lines with meet $M-M$ relationships are placed in one partition.
19:  end if
20: end for
21: for each partition derived at Step-3 do {}
22:   if $PSR < PS$ then
23:     Add lines from partition to Outlier partition
24:   end if
25: end for
26: File_Prune() {Input is the collection of newly created partitions}
27: Return() {Output is collection of pruned new partitions}
```

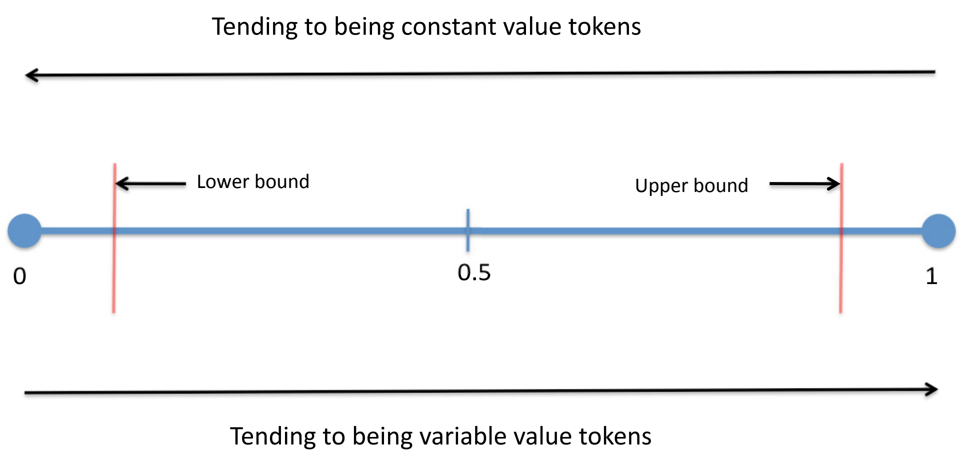

Tending to being constant value tokens
Lower bound
Upper bound
0
0.5
1
Tending to being variable value tokens

Figure 3.8: Deciding on how to treat 1-M and M-1 relationships. This procedure is implemented in the Get_Rank_Position function.

3.3.4 Step 4: Discover message type descriptions from each partition.

At this step of the algorithm, partitioning is complete and it is assumed that each partition represents a collection of messages produced by the same message type. The message type descriptions are determined by counting the number of unique tokens in each token position of a partition. If a token position has only one value then it is considered a constant value in the message type description. Token positions with more than one unique token are considered variables. This process is illustrated in Fig. 3.9.

Since the goal is to find all message types which may exist in an event log or ensure that the presence of every message type contained in an event log is reflected in the message types produced, the possibility of "*outliers*" interfering with the templates produced at this step is not of major concern. Hence, we set the threshold for determining a variable token position as any token position with more than one unique token.

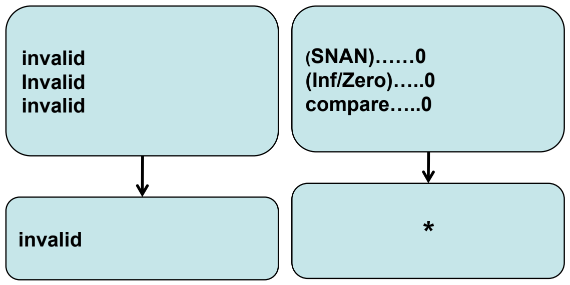


Figure 3.9: IPLoM Step-4: Discover message type descriptions. If the cardinality of the unique token values in a token position is equal to 1, then that token position is represented by that token value in the template, else the token position is represented with an "*".

3.3.5 Algorithm Parameters

In this section, an overview of the parameters used by IPLoM is given. The fact that IPLoM has several parameters (which can be used to tune its performance) provides flexibility for the system administrators. This gives them the option of using their expert knowledge when they deem it necessary.

- **File Support Threshold:** Ranges between [0,1]. It reduces the number of partitions produced by IPLoM. Any partition whose instances have a support value less than this threshold is discarded. The higher this value is set, the fewer the number of partitions will be produced. This parameter is similar to the *support threshold* defined for SLCT and Loghound.

- **Partition Support Threshold:** Ranges between [0,1]. It is a threshold which controls backtracking. Based on the experiments, the guideline is to set this parameter to very low values, i.e. < 0.05, for optimum performance.

- **Upper_Bound and Lower_Bound:** Ranges between [0,1]. They control the decision on how to treat the M side of relationships in Step-2. Lower_Bound should usually take values < 0.5 while Upper_Bound takes values > 0.5.

- **Cluster Goodness Threshold:** Ranges between [0,1]. It is used to avoid further partitioning. Its optimal setting is in the range of $0.3 - 0.6$.

3.4 Evaluation and Results

The design goal for IPLoM was threefold. The first was to design an algorithm which is able to find all message types which may exist in a given log file. The second was to give every message type an equal chance of being found irrespective of the frequency of its instances in the data. The third was to design an algorithm which will produce message types at an abstraction level preferred by a human observer. Therefore, the discussions in this section will begin by first describing the setup of the experiments in Section 3.4.1 and then providing results, in Section 3.4.2, which show how the goals listed were met when IPLoM is utilized with its primary goal: finding all message types. Resource consumption statistics (CPU and Memory) for the SLCT, Loghound and IPLoM are provided as well in Section 3.4.2. The discussions in sections 3.4.3 and 3.4.4 show how varying the line support threshold (FST) using low absolute counts and percentage values, respectively affects the results of the algorithms. SLCT, Loghound and Teiresias need a line support threshold to extract message types, while IPLoM does not. For SLCT and Loghound, this support value can be specified either as a percentage of the number of events in the event log or as an absolute value. For this reason, two sets of experiments were run using support values specified as percentages and as absolute values. In either case, the support values were set to low values because intuitively this allows for finding most of the message types in the data, which is one of the desired goals. In section 3.4.5, parameter sensitivity analysis results are presented. In section 3.4.6 an experiment is presented using logs from one of the world's fastest supercomputers that suggests that IPLoM may have linear complexity. In section 3.5 the performance limits of IPLoM are discussed.

3.4.1 Experimental Setting

All experiments were run on an iMac7 desktop computer running Mac OS X 10.5.6. The machine has an Intel Core 2 Duo processor with a speed of 2.4GHz and 2GB of memory. IPLoM's performance was tested against those of SLCT, Loghound and Teiresias. The four algorithms were tested against seven log datasets which were obtained from different sources; Table 4.4 gives an overview of these datasets. The HPC log file is a publicly available data-set collected on high performance clusters

Table 3.1: Log Data Statistics

Name	**Description**	**# Messages**	**# Msg. Types**
HPC	High Performance Cluster (Los Alamos)	433,490	106
Syslog	OpenBSD Syslog	3,261	60
Windows	Windows Oracle Application Log	9,664	161
Access	Apache Access Log	69,902	14
Error	Apache Error Log	626,411	166
System	OS X Syslog	24,524	9
Rewrite	Apache mod_rewrite Log	22,176	10

at the Los Alamos National Laboratory NM, USA [39]. The Access, Error, System and Rewrite data-sets were collected on the faculty network at Dalhousie University, while the Syslog and Windows files were collected on servers owned by a large ISP. Due to privacy issues, the Dalhousie and ISP datasets cannot be made available to the public.

The message type descriptions of these 7 data-sets were produced manually by Tech Support members of the Dalhousie Faculty of Computer Science. Table 4.4 gives the number of message types identified in each file manually. Some form of automation (e.g. regular expressions) could have been utilized in the labeling process, however the decision on what constitutes a message type was completed manually by system administrators of Dalhousie Computer Science. Again, due to privacy issues, the manually produced message type descriptions are provided only for the HPC dataset [1]. These cluster descriptions became the gold standard against which to measure the performance of the algorithms as an information retrieval (IR) task. As in classic IR, the performance metrics were Precision, Recall and F-Measure, which are described by Eqs. 3.3, 3.4 and 3.5, respectively. The terms TP, FP and FN in the equations are the number of True Positives, False Positives and False Negatives, respectively. Their values are derived by comparing the set of manually produced message type descriptions to the set of templates extracted by each algorithm. In the evaluation, a message type description is still considered a FP even if it matches a manually produced message type description to some degree: the match has to be exact for it to be considered a TP.

For completeness the Precision, Recall and F-Measure values were evaluated using

[1]Descriptions are available for download from http://web.cs.dal.ca/~makanju/iplom

three different methods. In two methods, the results of the algorithms were evaluated as a classification problem, using the manually produced message types as classes. In this case, the correlation of the automatically assigned labels to the manually produced labels was evaluated. This classification evaluation produced Micro-average and Macro-average results. These results are referred to as *"Micro"* and *"Macro"*. In the third method, the manually produced message type descriptions are compared against the automatically produced ones. This evaluation method is called "IR". The "IR" evaluation method satisfies the intended goals better, as it tests the accuracy of the message types extracted. The "IR" method evaluates the results based strictly on how well the message types extracted match the manually produced message types.

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.5}$$

The parameter values used in running the experiments which produced the baseline results and the results in sub-sections 3.4.3 and 3.4.4 are provided in Table 3.2. The seed value for SLCT and Loghound is a seed for a random number generator used by the algorithms; all other parameter values for SLCT and Loghound are left at their default values. Similarly, the parameters for Teiresias were chosen to achieve the lowest support value allowed by the algorithm. The IPLoM parameters were all set empirically except in the case of the cluster goodness threshold and the partition support threshold.

In setting the cluster goodness threshold, IPLoM was tested against the HPC file while varying this value. The parameter was then set to the value (0.34) which gave the best result and was kept constant for the other files used in the experiments carried out. On the other hand, the partition support threshold was set to 0 to provide a baseline performance. Such a setting for the performance threshold implies that no backtracking was done during partitioning.

It is pertinent to note that the Teiresias algorithm could not be tested against all datasets. This was due to its inability to scale to the size of the datasets. This is a problem which is attested to by other researchers [68]. Thus, in this work, Teiresias could be tested only against the Syslog dataset. The memory consumption results were obtained by monitoring the processes for each algorithm by using the Unix *ps* and *grep* utilities.

Table 3.2: Algorithm Parameters

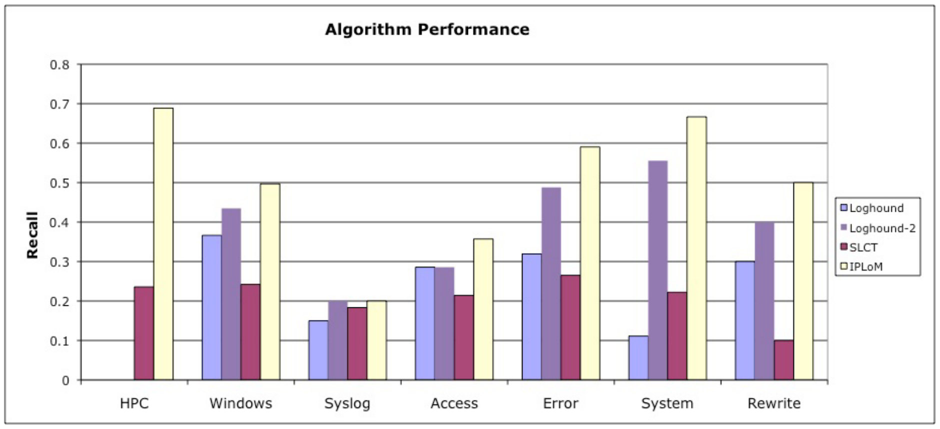
Algorithm	**Parameter**	**Value**
SLCT and Loghound		
	Support Threshold (-s)	0.01 - 0.1
	Seed (-i)	5
Teiresias		
	Sequence Version	On
	L (min. no. of non wild card literals in pattern)	1
	W (max. extent spanned by L consecutive non wild card literals)	15
	K (Min. no. of lines for pattern to appear in)	2
IPLoM		
	File Support Threshold (Percentage)	0 - 0.1
	File Support Threshold (Absolute)	1 - 20
	Partition Support Threshold	0
	Lower Bound	0.1
	Upper Bound	0.9
	Cluster Goodness Threshold	0.34

3.4.2 Baseline Experiments

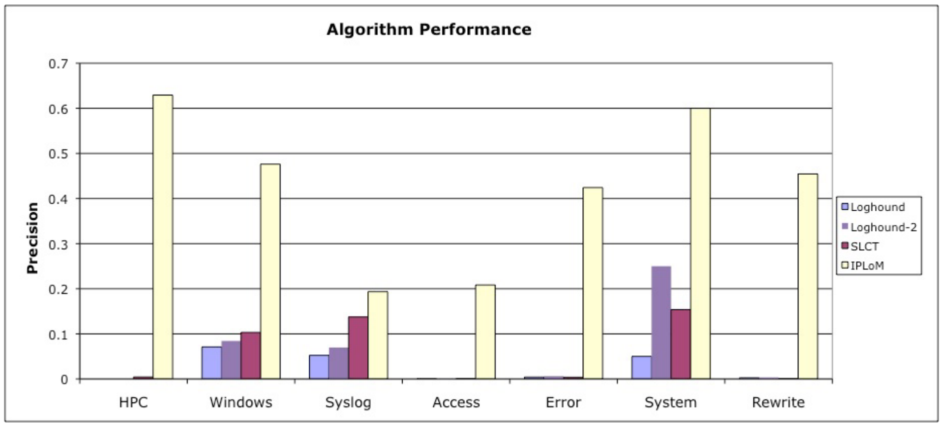
The result of the default evaluation of SLCT, Loghound and IPLoM is shown in Figure 3.10. The graphs in Figure 3.10 show the results for the Recall, Precision and F-Measure metrics for all the algorithms using the “IR” evaluation method. Since one of the goals of IPLoM is to find all message types which may exist in a log file, this set of experiments was run with the lowest file support threshold possible, which is an absolute support value of *1*. SLCT and Loghound would not work efficiently with an absolute support value of *1*, so they were run with *2* instead. An absolute support value of *1* means every line/word will be considered frequent and the result of the algorithms will be reduced to the case of finding unique lines for SLCT or the case of finding all possible templates for Loghound. Both situations are not desirable. Since Teiresias worked only on the Syslog data-set, its results are not included in the analysis. Utilizing the parameter values listed in Table 3.2, Teiresias produced a Recall performance of 0.1, a Precision performance of 0.04, which led to an F-Measure performance of 0.06 using the IR evaluation method. By providing "IR" evaluations which compare the results of the algorithms with manually produced results, we evaluated how well the third design goal has been met, which was to design an algorithm which will produce message types at an abstraction level preferred by a human observer.

For fairness in comparison, two evaluations for Loghound are provided. Loghound, being a frequent itemset mining algorithm, is unable to detect variable parts of a message type when they occur at the tail end of a message type description. For example, if it is intended to find the message type description *" Error code: *"*, it is possible for Loghound to find the message type description as *"Error code:"*, without the trailing variable at the end. In such a situation, Loghound would not be credited with finding the message type description. Therefore, a second set of evaluations for Loghound (referred to as Loghound-2) was generated. This evaluation adjusts Loghound's results by comparing them to manually produced message types when the last trailing variables are discarded.

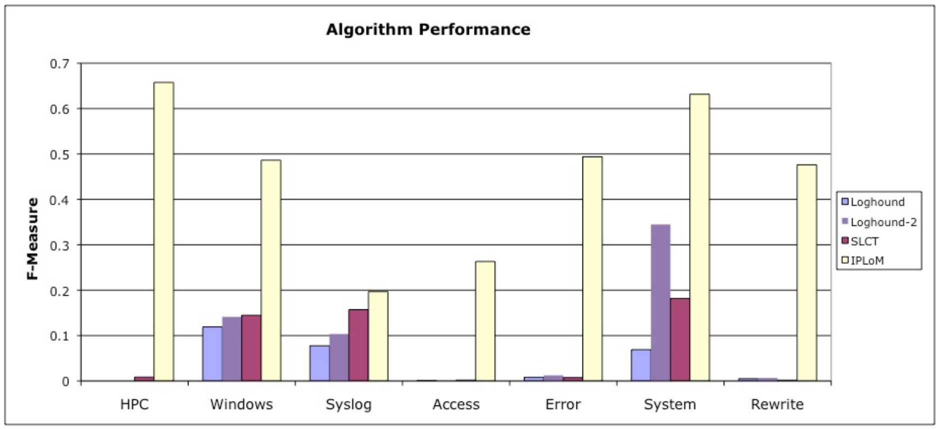
However, these results are for information purposes only. Considering message type descriptions where the number of trailing variables cannot be assessed is detrimental to the goal of finding only meaningful message types at an abstraction level preferred by a human observer. When the number of trailing variables in a message type description cannot be assessed, the event size is unknown. The "Event Size" is a means of differentiating between messages types. This leads to a loss of meaning and ambiguity. Consider these three actual examples of message types found manually: *"Link *", "Link error on broadcast tree", "Link in reset"*. Without the trailing variable the first message type becomes *"Link"* which can be interpreted to mean any message which starts with the word "Link", since the event size of the message type is unknown. This interpretation means that an instance of the first message type cannot be distinguished from an instance of the second or third message types. Consequently, even though we have presented Loghound-2 results where the trailing variables are not used, we believe that in practice the trailing variables are needed (Loghound results below).



(a) Recall



(b) Precision



(c) F-Measure

Figure 3.10: Comparing algorithm IR performances at lowest support values.

Table 3.3: Log Data Event Size Statistics

Name	**Min**	**Max**	**Avg.**
HPC	1	95	30.7
Syslog	1	25	4.57
Windows	2	82	22.38
Access	3	13	5.0
Error	1	41	9.12
System	1	11	2.97
Rewrite	3	14	10.1

The average IR F-Measure performance across the data-sets, at this default support level, is 0.07, 0.04, 0.10 and 0.46 for SLCT, Loghound, Loghound-2 and IPLoM respectively. However, as stated in [18], in cases where data sets have relatively long patterns or low minimum support thresholds have been used, apriori-based algorithms incur non-trivial computational costs during candidate generation. The event size statistics for the datasets as outlined in Table 3.3 show the *HPC* file as having the largest maximum and average event size. Loghound was unable to produce results on this data set with an absolute support value of 2. The algorithm crashed due to the large number of item-sets which had to be generated, as can be seen in Figure 3.10. However, this was not a problem for SLCT (as it generates only 1-itemsets). These results show that Loghound is vulnerable to the computational cost problems outlined in [18], which is not a problem for IPLoM as its computational complexity is not affected adversely by long patterns or low minimum support thresholds. In terms of performance based on the event size, Table 3.4 shows consistent performance from IPLoM irrespective of event size, while SLCT and Loghound seem to suffer with mid-size clusters. Evaluations of Loghound considering the trailing variable problem shows Loghound achieving its best results for message types with a large event size and achieving results which are comparable to IPLoM in the other categories.

The ability to discover message types in event logs irrespective of how frequently its instances appear in the data is another cardinal goal in the design of IPLoM. The performance of the algorithms using this evaluation criterion is outlined in Table 3.5.

Table 3.4: Algorithm performance based on cluster event size

Event Size	No. of Clusters	Percentage Retrieved(%)			
		SLCT	Loghound	Loghound-2	IPLoM
1 - 10	316	12.97	13.29	49.68	53.80
11 - 20	142	7.04	9.15	35.92	49.30
>21	68	15.15	16.67	77.27	51.52

The results show a reduction in performance for all the algorithms for message types with a few instances, however IPLoM's performance was more resilient.

The resource consumption results for SLCT, Loghound and IPLoM are presented in Tables 3.6, 3.7 and 3.8. The tables show results for CPU, virtual memory and resident memory consumption, respectively. It is noted that the results for Loghound on the HPC data set are results collected before its process crashed. However, the statistics for this case show why the process crashed. Its virtual memory and resident memory consumption had gone up to ~4GB and ~1.6GB, respectively. This is for a file which is ~11.4MB on disk. These results corroborate the initial assertion on the failure of Loghound on this data set, i.e. a large number of itemsets having been generated at a low support value. Furthermore, the CPU consumption does not give a true picture of the time before the algorithm crashed, the actual time between the start of the process and its crashing was 62 mins. The *ps* utility measures the CPU time consumed by the process alone. In this case, all the CPU time spent by the OS in performing swaps between resident and virtual memory as a result of the process was not recorded.

Table 3.5: Algorithm performance based on cluster instance frequency (*LH* $\rightarrow$ *Loghound*)

Instance Frequency Range	# Clusters	Percentage Retrieved(%)			
		SLCT	LH	LH-2	IPLoM
1 - 10	263	2.66	1.90	38.02	44.87
11 - 100	144	16.67	18.75	50.69	47.92
101 - 1000	68	20.59	23.53	63.24	72.06
>1000	51	34.00	38.00	74.00	82.00

In spite of the fact that the implementation of IPLoM uses no sophisticated memory management techniques[2], it produces results which are comparable to those of SLCT and Loghound in terms of both virtual memory and resident memory consumption. The memory consumption of IPLoM can be improved further by using optimization algorithms and data structures. In terms of CPU time, however, its results are only comparable to those of SLCT and Loghound for the smaller data sets, i.e. Rewrite, Syslog, System and Windows. This is not seen as a problem since IPLoM is designed for offline extraction of message types. Having said this, we believe there is still a lot of room for improvement in CPU time consumption for the following reasons.

- The most important factor in determining the processing time for IPLoM is time spent scanning the database. This suggests that IPLoM may have linear complexity. IPLoM was implemented for research purposes, so each step of the algorithm was implemented separately. This implies that there was no information sharing between the steps of the algorithm. This led to unnecessary scans of the database for information gathering, which could have been avoided.
- IPLoMs partitioning of the data is in effect a decomposition of the message type extraction problem. This makes IPLoM a good candidate for parallel processing.

3.4.3 Absolute Support Values

In this set of experiments the results using absolute support values in the range of $1 - 20$ are compared, with a minimum of 2 for SLCT and Loghound. The goal here, as with the experiments in Section 3.4.4, is to determine how varying the file support threshold within a range of low values affects the performance of the algorithms. As stated earlier using low file support values ensures a good chance of finding all the message types in the file, which is one of the aforementioned goals.

The average F-Measure results using the “Micro”, “Macro” and “IR” evaluations for SLCT, Loghound and IPLoM are highlighted in Table 3.9. The results show

[2]SLCT and Loghound utilize string hashing functions and cache trees for efficient memory utilization [76, 77]

Table 3.6: CPU Time in Minutes

	SLCT	Loghound	IPLoM
HPC	<1.0	3.5	2.0
Syslog	<1.0	<1.0	<1.0
Windows	<1.0	<1.0	<1.0
Access	<1.0	<1.0	5.4
Error	<1.0	<1.0	37.5
System	<1.0	<1.0	<1.0
Rewrite	<1.0	<1.0	<1.0

Table 3.7: Maximum Virtual Memory Consumption in KBs

	SLCT	Loghound	IPLoM
HPC	608,348	4,186,284	99,248
Syslog	600,284	600,284	98,992
Windows	601,232	601,056	98,992
Access	614,088	615,760	101,040
Error	623,504	613,964	101,040
System	599,636	600,032	98,992
Rewrite	600,032	600,032	98,992

that IPLoM performs better than the other algorithms on all data sets in the IR evaluation, which measures the goodness of the clusters produced. In the Micro and Macro evaluations, IPLoM does even better than the other algorithms in general. However, there are performance improvements for SLCT and Loghound and in one case (with the Syslog dataset) SLCT actually performs better than IPLoM.

Table 3.8: Maximum Resident Memory Consumption in KBs

	SLCT	Loghound	IPLoM
HPC	10,388	1,619,156	19,196
Syslog	1,260	1,260	19,008
Windows	2,660	2,976	19,072
Access	15,688	22,748	21,136
Error	24,900	20,968	21,056
System	320	820	18,988
Rewrite	920	1,252	19,084

Table 3.9: Average F-Measure performance of algorithms using absolute support values

F-MEASURE PERFORMANCE						
	HPC			Syslog		
	SLCT	**Loghound**	**IPLoM**	**SLCT**	**Loghound**	**IPLoM**
Micro	0.64	0.55	0.66	0.10	0.06	0.07
Macro	0.25	0.45	0.45	0.13	0.07	0.11
IR	0.02	0.01	0.59	0.14	0.08	0.14
	Windows			**Access**		
	SLCT	**Loghound**	**IPLoM**	**SLCT**	**Loghound**	**IPLoM**
Micro	0.22	0.25	0.28	0.00	0.00	0.00
Macro	0.17	0.18	0.22	0.11	0.15	0.20
IR	0.18	0.11	0.34	0.00	0.00	0.26
	Error			**System**		
	SLCT	**Loghound**	**IPLoM**	**SLCT**	**Loghound**	**IPLoM**
Micro	0.68	0.28	0.82	0.20	0.16	0.83
Macro	0.22	0.17	0.31	0.18	0.09	0.56
IR	0.01	0.01	0.43	0.15	0.07	0.75
	Rewrite					
	SLCT	**Loghound**	**IPLoM**			
Micro	0.08	0.05	0.83			
Macro	0.10	0.13	0.30			
IR	0.01	0.01	0.49			

3.4.4 Percentage-based Support Values

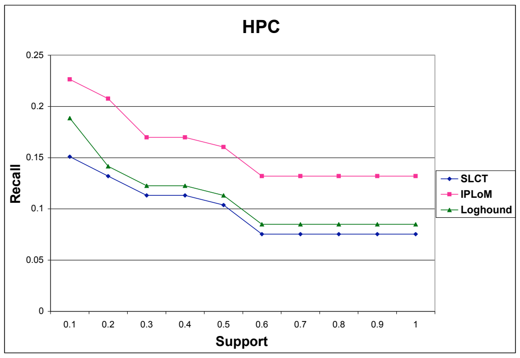
A system administrator can specify a support value using an absolute value (as in the section above) or a value which is dependent on the number of lines in the event log, i.e. a percentage. To determine IPLoMs performance when support values are specified in this way another set of experiments using percentage-based support values was performed. For the same reasons as described above, the range of values is low, i.e. 0.1% - 1.0%. The F-Measure results of this scenario show IPLoM performing better than the other algorithms on all the tasks. A single factor ANOVA test performed at 5% significance on the results shows a statistically significant difference in all the results except in the case of the Syslog file. Detailed results of the ANOVA tests can be found in Appendix D.

The rest of the results for this set of experiments were evaluated using the IR method. A detailed summary of the Recall, Precision and F-Measure results can be found in Figs. 3.11, 3.12 and 3.13, respectively. The results show that IPLoM outperforms both algorithms in all cases.

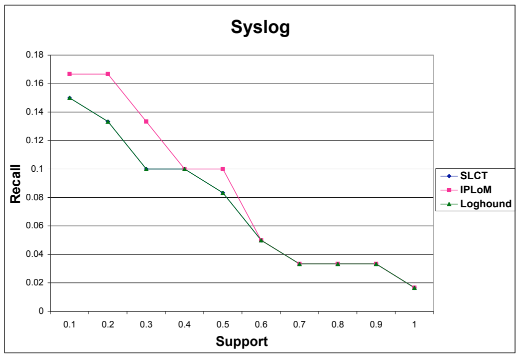
3.4.5 Parameter Sensitivity Analysis

IPLoM has five parameter values which can affect its results. These parameters are the File Support Threshold (FST), Partition Support Threshold (PST), Cluster Goodness Threshold (CGT) and the Upper Bound (UB) and Lower Bound (LB) thresholds used to decide if the “many” end of a 1-M relationship represents constant values or variable values. It is important that the sensitivity of IPLoM’s performance to the value settings of these parameters is assessed. In this section such an analysis is presented. IPLoM was tested against the datasets using a wide range of values as outlined in Table 3.10. Since the FST used in IPLoM is similar to the support threshold used in SLCT and Loghound they were tested as well using the range of values for FST in Table 3.10.

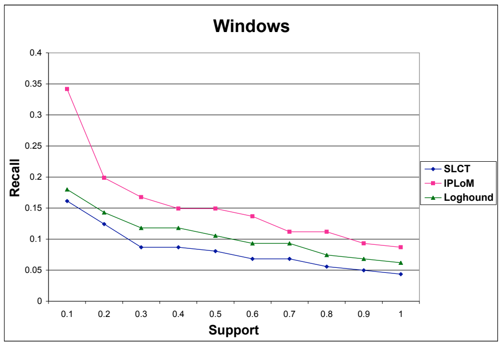
The results show that IPLoM is most sensitive to varying values of FST as can be seen in Fig. 3.14. This can be explained by the observation that increasing the support value decreases the number of event types which can be found, since any event type with instances which fall below the support value cannot be found. However, the graphs show that generally, for support values greater than 20%, there is not



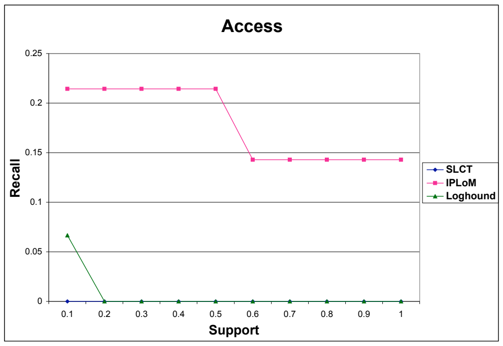
(a) HPC



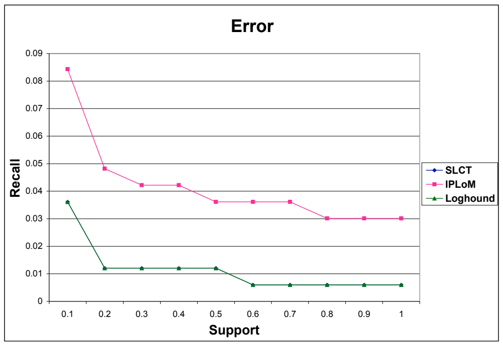
(b) Syslog



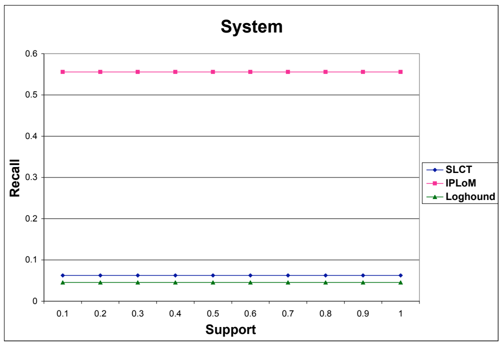
(c) Windows



(d) Access

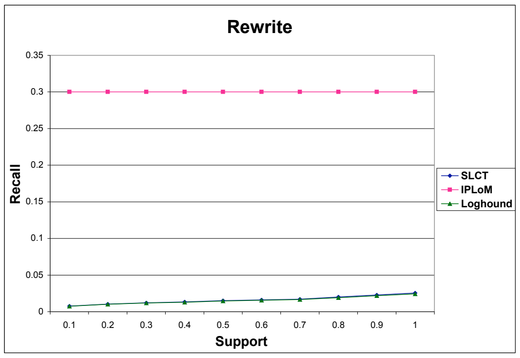


(e) Error



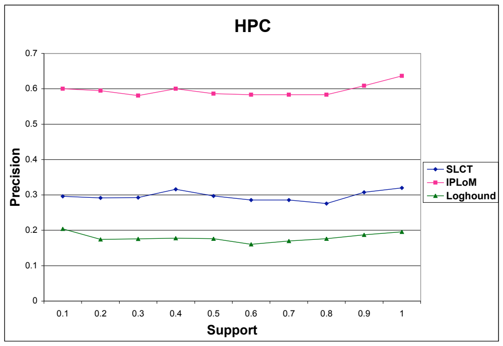
(f) System

Figure 3.11: Comparing Recall performance of IPLoM, Loghound and SLCT using support values.

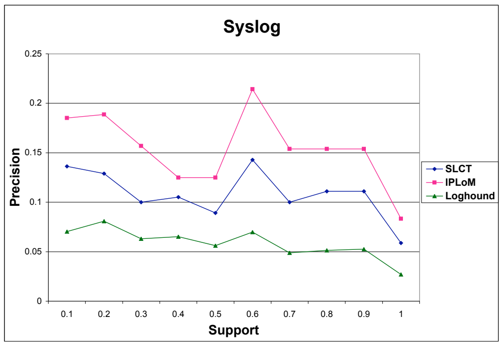


(g) Rewrite

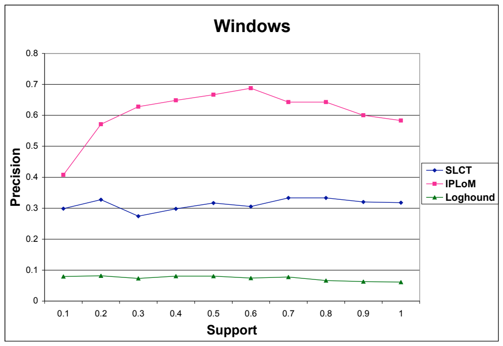
Figure 3.11: Comparing Recall performance of IPLoM, Loghound and SLCT using support values.



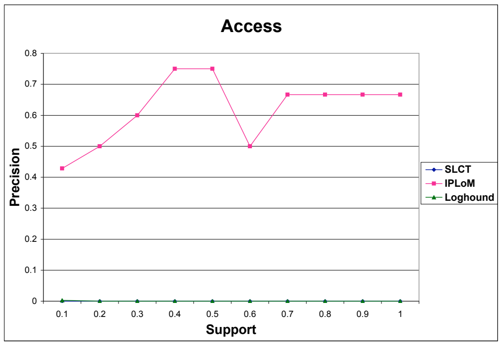
(a) HPC



(b) Syslog

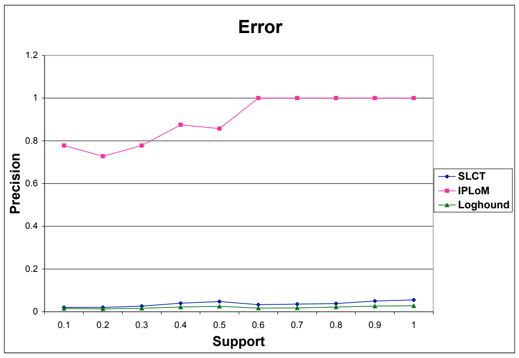


(c) Windows

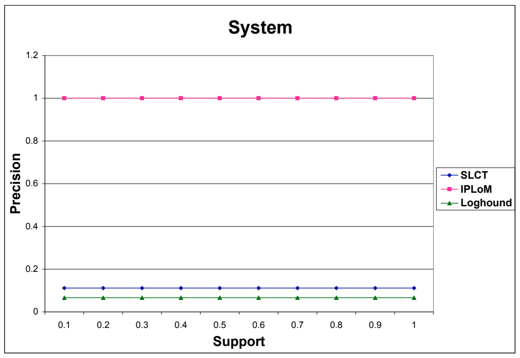


(d) Access

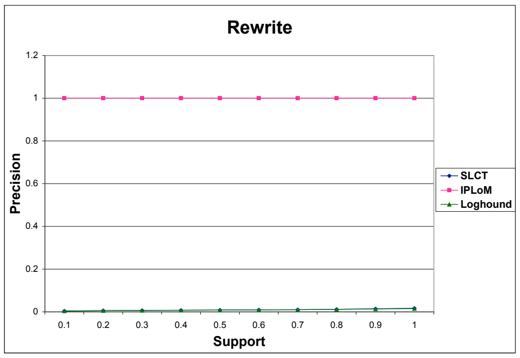
Figure 3.12: Comparing Precision performance of IPLoM, Loghound and SLCT using support values.



(e) Error

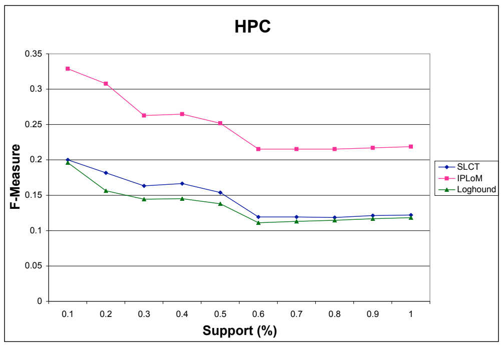


(f) System

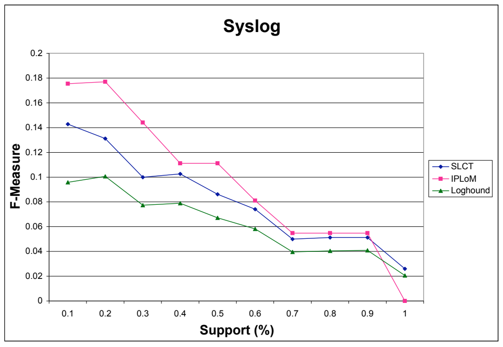


(g) Rewrite

Figure 3.12: Comparing Precision performance of IPLoM, Loghound and SLCT using support values.

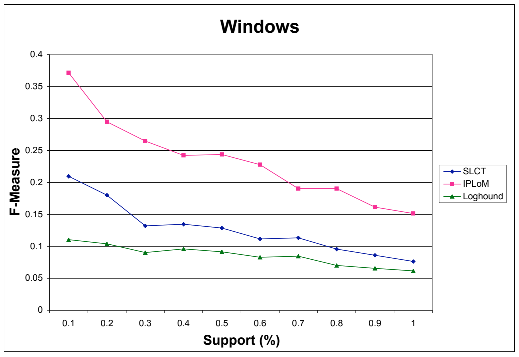


(a) HPC

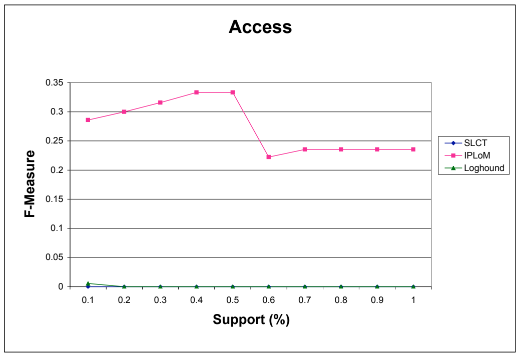


(b) Syslog

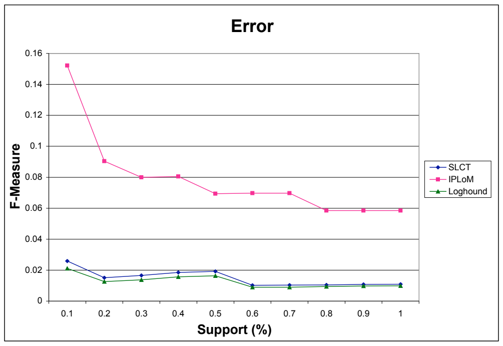
Figure 3.13: Comparing F-Measure performance of IPLoM, Loghound and SLCT support values.



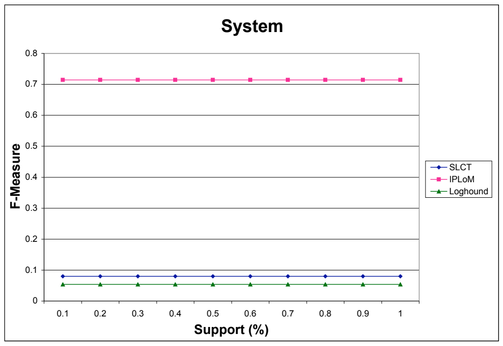
(c) Windows



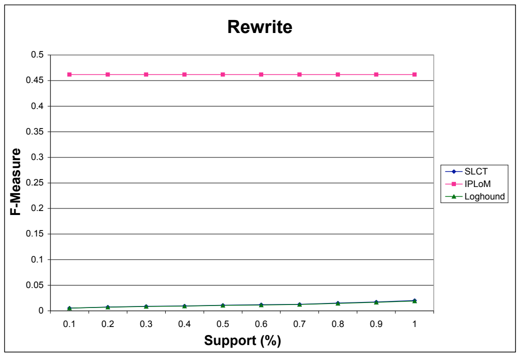
(d) Access



(e) Error



(f) System



(g) Rewrite

Figure 3.13: Comparing F-Measure performance of IPLoM, Loghound and SLCT support values.

much difference in the performance of the algorithms. Using the standard deviation over the range of results for each parameter, as seen in Table 3.11, the sensitivity of the algorithms to changing parameter values can be evaluated. The results show that IPLoM is stable in the face of changing parameter values. The largest standard deviation values are found with IPLoM under the FST parameter, which is due to IPLoM's superior performance for FST values less than 20%.

3.4.6 Complexity

The results observed in this section suggest that the computational complexity of IPLoM is linear with respect to the number of events in any log. The experiments were performed using the BlueGene/L (BGL) dataset.

The BlueGene/L data set is a publicly available high performance computing log dataset [75]. The BlueGene/L supercomputer is a well known HPC machine designed by IBM. It is located at Lawrence Livermore National Labs (LLNL) in Livermore, CA, USA. According to the Top-500 supercomputing site BlueGene/L ranked #5 in its list of the fastest supercomputers in the world [73] in January 2011. The BGL data contains 4.7M event log entries from BlueGene/L covering a 215 day period. The size of the event log file on disk is 1.2GB.

The experiment was carried out by running IPLoM against portions of the BGL data which differ in size by units of $100,000$ events. IPLoM was tested against each portion 10 times and the time for completion was recorded for each run. The graph in Fig.3.15 records the average time over the 10 runs and the amount of time required to run each unit of $100,000$ events.

Table 3.10: Parameter Value Ranges Used for Sensitivity Analysis

Parameter	**Range**
File Support Threshold(%)	0 - 100
Partition Support Threshold(%)	0 - 5
Lower Bound	0.1 - 0.5
Upper Bound	0.5 - 0.9
Cluster Goodness Threshold	0 - 1

The results suggest that IPLoM is a lightweight algorithm with linear complexity in terms of the size of an event log, for the data sets we experimented with. IPLoM's complexity in the general case is a topic of future research. This result provides strong incentive for future work where the complexity of IPLOM can be explored analytically.

3.5 Analysis of Performance Limitations

The IPLoM algorithm, as with all algorithms which utilize heuristics, is capable of making errors and does in fact make errors during its partitioning phase. An analysis of these errors is given in this section. In addition, likely remedies are described where possible.

3.5.1 Insufficient Information in Data

Apart from the message type descriptions produced by all the algorithms as output, IPLoM has the added advantage of producing the partitions of the log data which represent the actual message types. This provides two sets of results which can be evaluated for IPLoM, the partitions and their descriptions. While evaluating the partition results of IPLoM it was discovered that in certain cases it was impossible for IPLoM to produce the right message type description for a partition due to the fact that the partition contained only one event line or contained identical event lines. This situation would not pose a problem for a human subject who would be able to use semantic and domain knowledge to determine the right message type description. This problem is illustrated in Fig. 3.16. This indicates that the IR comparison of the message type descriptions produced by IPLoM does not give a complete picture of

Table 3.11: Standard Deviation over F-Measure results for parameter values

	FST			**PST**	**CGT**	**LB:UB**
	Loghound	**SLCT**	**IPLoM**	**IPLoM**	**IPLoM**	**IPLoM**
HPC	0.017	0.017	0.197	0.009	0.107	0.048
Syslog	0.025	0.000	0.058	0.000	0.013	0.025
Windows	0.036	0.044	0.146	0.002	0.088	0.018
Access	0.000	0.001	0.079	0.036	0.085	0.006
Error	0.005	0.010	0.146	0.034	0.034	0.028
System	0.035	0.066	0.279	0.000	0.000	0.197
Rewrite	0.001	0.000	0.123	0.000	0.000	0.000

IPLoM's performance. To get a complete picture of IPLoM's capabilities, IPLoM's performance based on partitioning results was evaluated. These results are called *Partitions* in Fig. 3.17, while the message type description results are called *Before*. The partition comparison differs from the message type description results because cases where IPLoM came up with the right partition but was unable to come up with the right cluster description were marked as correct. The results show an average F-Measure of 0.48 and 0.78 for IPLoM when evaluating the results of IPLoM's message type description output and partition output, respectively. Similar results are noticed as well for Precision and Recall.

Due to the fact that SLCT and Loghound do not generate partitions to evaluate (however, these partitions can be found through post-processing if desired) and since it can be argued that the "insufficient information in data" scenario could apply to them as well, another experiment was constructed. In this case, counter-examples for all the cases in which there was insufficient information in the event data for the algorithms to come up with the message type descriptions which were inserted into the data. Subsequently, SLCT, Loghound and IPLoM were run against the new data sets with counter-examples inserted. SLCT and Loghound were run in this case with the absolute support values, which gave their best results in the experiments described in Section 3.4.3 above while IPLoM was run in its default state. These results are called *After* in Table 3.17. The results show that unlike SLCT and Loghound, IPLoM was able to make use of the new information to improve its results in all cases. These results show that IPLoM can achieve an average Recall of 0.83, Precision of 0.74 and F-Measure of 0.78.

3.5.2 Ambiguous Token Delimiters

In the problem of message type extraction the assumption is that the space character acts as the token delimiter. Close inspection of certain messages in the log files ascertains that this assumption is not true in all cases. The most common example occurs when part or all of a message contains a *"variable = value"* phrase. In some cases there is no space character between the *variable* token and the = sign and also between the *value* token and the = sign. This scenario becomes a problem for IPLoM when, for example, the log messages *"Temperature reading: ambient=30"*, *"Temperature* *reading: ambient=25"* and *"Temperature reading: ambient=28* are evaluated to the type *"Temperature reading: ambient=*"* by a human observer. When and if IPLoM produces a partition correctly containing these log messages the type produced will be *"Temperature reading: *"*, due to IPLoM's inability to separate the tokens in the *"variable = value"* phrase. An approach to mitigating this problem could involve scanning for words containing an = sign before message type extraction and splitting such words into three parts at the = sign. The word triple can then be concatenated at the end of the extraction process which will ensure that future instances of the message type can be matched to the message type description produced [68].

3.5.3 Clusters with events of variable size

Another scenario which occurs is with message types with variable sizes. IPLoM assumes that messages belonging to the same message type should have the same number of tokens or event sizes. Again, on close inspection, this is found to not to be true in some cases. Message types with variable sizes occur usually when a variable position in the description can contain strings instead of a single token. For example, consider the messages *"The LightScribe service has started"* and *"The Message Queuing service has started"* which should belong to the same message type and have the message type description *"The * service has started"*. These messages have a differing number of tokens and would be separated by Step-1 of IPLoM's partitioning process which likely would produce two message type descriptions for this cluster *"The * service has started"* and *"The * * service has started"*, the latter template being redundant.

This problem can be mitigated by performing message type description refinement after the message types are produced. An approach such as the string edit distance or Levenshstein distance used in [15], can be utilized for this step if necessary. The string edit distance provides a measure of distance between strings by counting the number of operations required to transform one string to another. Strings with a distance below a user-provided threshold can be merged. The templates of message types with variable size likely will not be very distant from each other using this approach. However, dealing with message types of variable size as a post-processing step rather than as part of the message type extraction process reduces the amount of computation required for the extraction process. In addition, it has the advantage of determining bounds on the range of event sizes for message types with variable size.

Generally, post-processing and refinement of the output from IPLoM is recommended for practical purposes. A hierarchy of message types and sub-types, using the variable tokens in a message type, the distance of message types from each other and user feedback, can be built at this stage as well, if required.

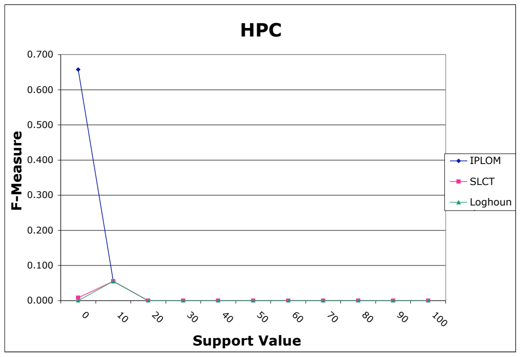
3.6 Discussion of Results

Message types are fundamental units in any application log file. Determining what message types can be produced by an application accurately and efficiently is therefore a fundamental step in the automatic analysis of log files. Once determined, message types provide not only groupings for categorizing and summarizing log data, which simplifies further processing steps such as visualization or mathematical modeling, but also a way of labeling the individual terms (distinct word and position pairs) in the data.

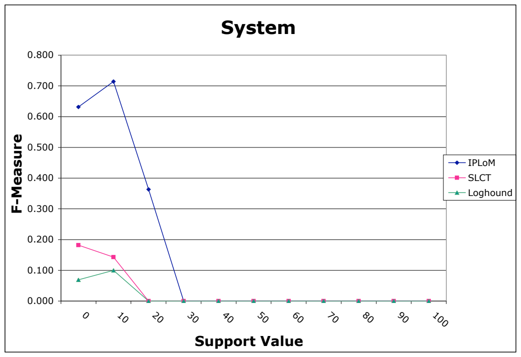
To date, there is no standard approach to tackling the problem of message type extraction in the literature [83]. In most cases message type extraction is done manually and never in an exhaustive manner. Usually, the task is reduced to what system administrators become familiar with over time and through experience. In IPLoM we have a lightweight algorithm which is able to find these message types automatically with an acceptable level of accuracy. IPLoM is able to find message types whether or not their instances are frequent in the log data. IPLoM's message type descriptions match human judgement very closely. Experimental evaluations show statistically significant better performance for IPLoM when compared to either SLCT or Loghound.

However, automatic analysis of event logs does not end with message type extraction. In IPLoM, a practical way of automatically extracting message types is available. In conjunction with the other fields in an event (host names, severity), message types can be used for more detailed analysis of log files. The rest of this thesis demonstrates this in practical terms. Examples of how message types can help improve the visualization, storage/retrieval of logs and modeling of event logs are demonstrated. As the focus of this thesis is fault management, a proof of concept for a framework for the automatic discovery of alert message types was designed.

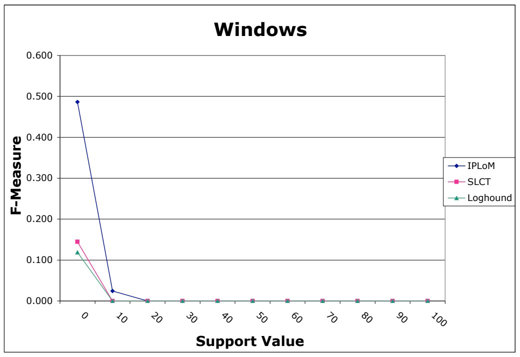
In most modern systems, this process still is done automatically through the use of a rule base which is built and maintained manually. The work in this thesis demonstrates how this entire process can be automated with minimal human input. The framework uses message types produced by IPLoM as its foundation.



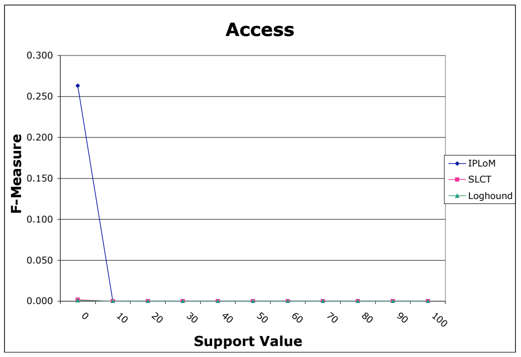
(a) HPC



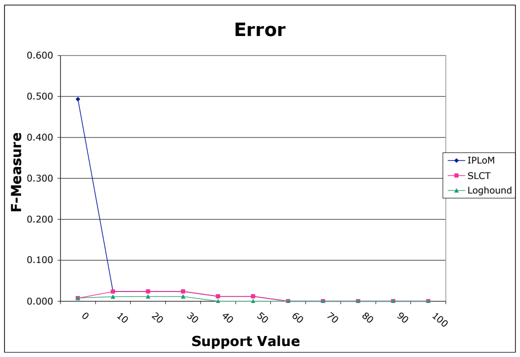
(b) Syslog



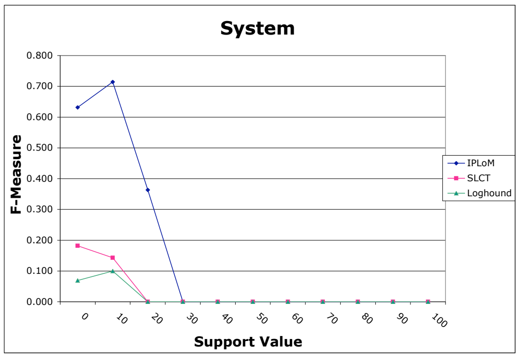
(c) Windows



(d) Access

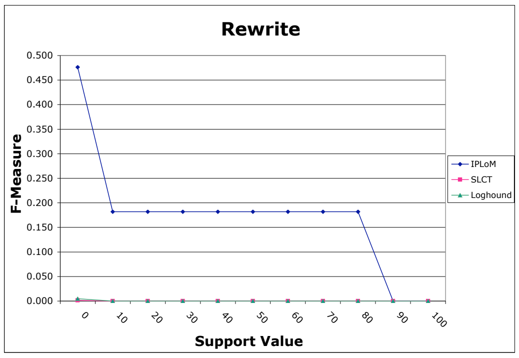


(e) Error



(f) System

Figure 3.14: F-Measure performance of IPLoM, Loghound and SLCT against FST values in the range 0% - 100%. 0% support values for SLCT and Loghound are equivalent to using an absolute support value of 2.



(g) Rewrite

Figure 3.14: F-Measure performance of IPLoM, Loghound and SLCT against FST values in the range 0% - 100%. 0% support values for SLCT and Loghound are equivalent to using an absolute support value of 2.

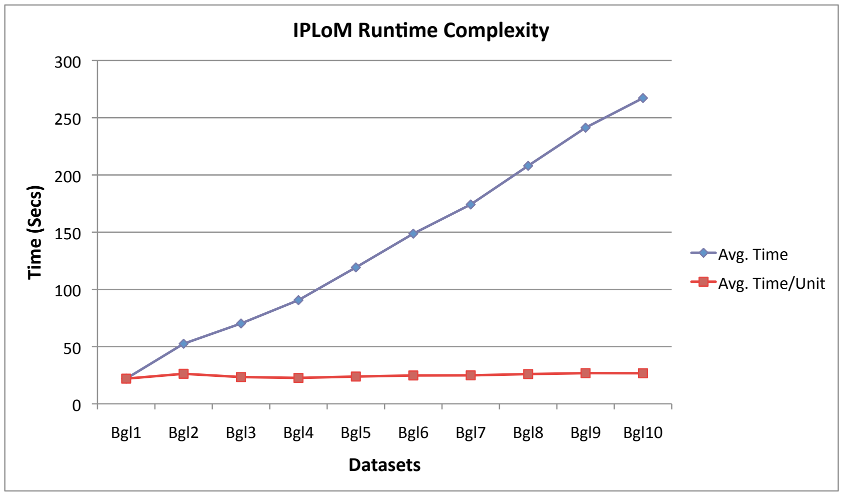


Figure 3.15: Runtime Cost of IPLoM on the BGL data. Each Bgl$x$ dataset contains $x * 100,000$ events.

%vspace-5mm

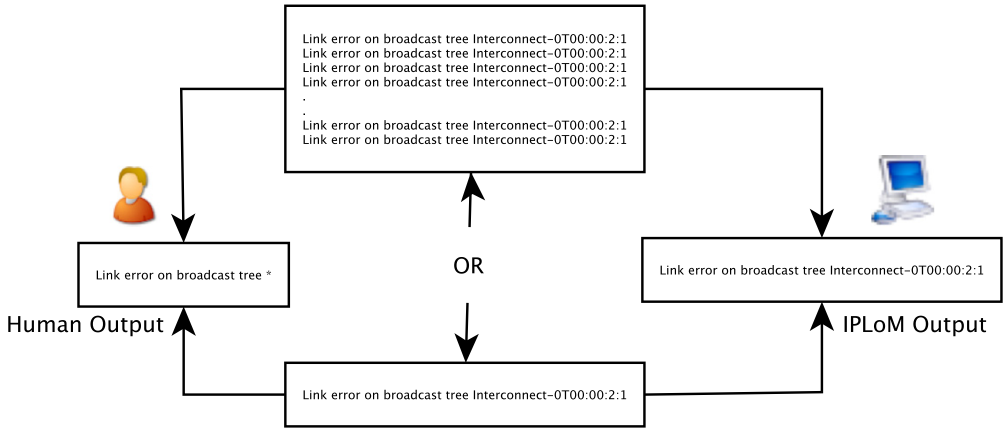
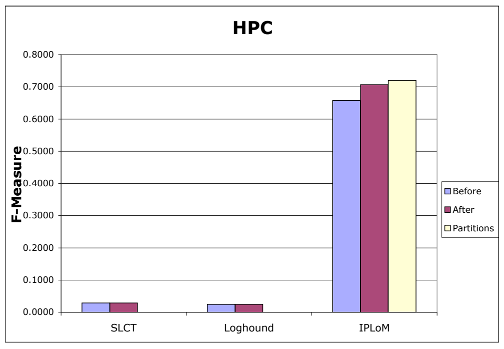
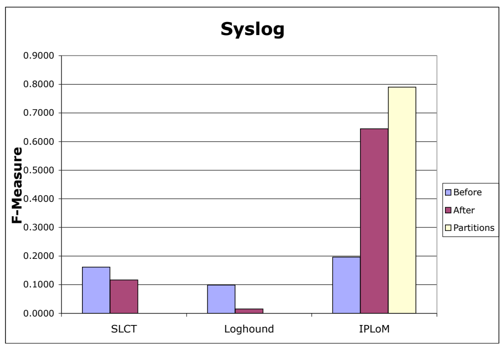

Link error on broadcast tree Interconnect–0T00:00:2:1
Link error on broadcast tree Interconnect–0T00:00:2:1
Link error on broadcast tree Interconnect–0T00:00:2:1
Link error on broadcast tree Interconnect–0T00:00:2:1
.
.
Link error on broadcast tree Interconnect–0T00:00:2:1
Link error on broadcast tree Interconnect–0T00:00:2:1
Link error on broadcast tree *
Human Output
OR
Link error on broadcast tree Interconnect–0T00:00:2:1
IPLoM Output
Link error on broadcast tree Interconnect–0T00:00:2:1

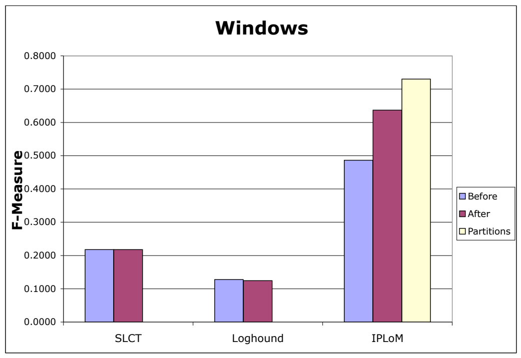
Figure 3.16: Example: Insufficient Information in Data.



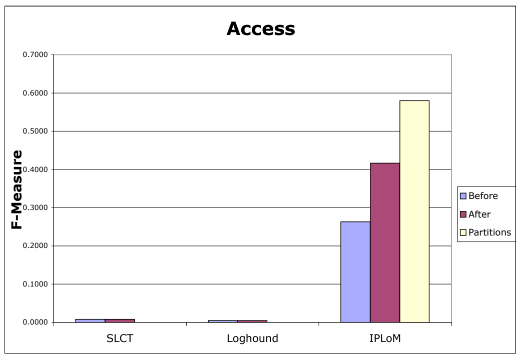
(a) HPC



(b) Syslog

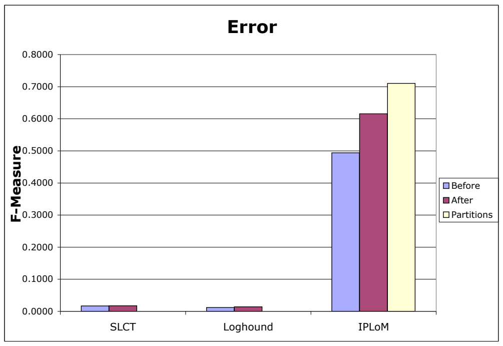


(c) Windows

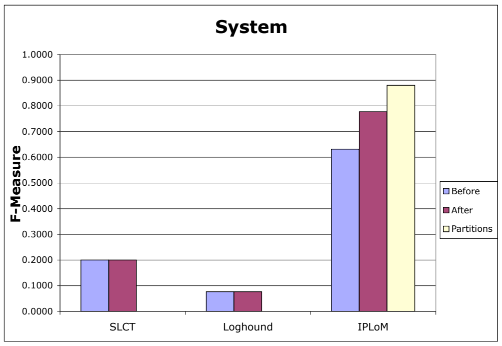


(d) Access

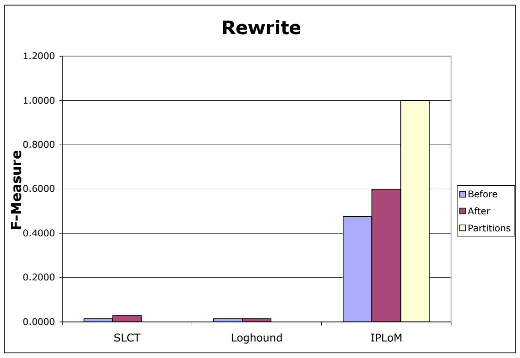
Figure 3.17: Comparing F-Measure performance of IPLoM, Loghound and SLCT before insertion of counter-examples, after insertion of counter examples and evaluating the accuracy of cluster partitioning before the insertion of counter examples.



(e) Error



(f) System



(g) Rewrite

Figure 3.17: Comparing F-Measure performance of IPLoM, Loghound and SLCT before insertion of counter-examples, after insertion of counter examples and evaluating the accuracy of cluster partitioning before the insertion of counter examples.

Chapter 4

Applications of Message Types in Event Log Management

Message types are the basic *concepts* in event logs, therefore their use in the management and analysis of event logs should not be treated as an option but as a necessity. To motivate the argument, an analogy from *Information Retrieval* and *Text/Document Processing* is drawn. These computer science fields deal with the processing of unstructured textual data, just as is the case with event logs. Consequently, event log analysis and management could borrow from techniques and concepts which have previously been successful in these fields.

As *Information Retrieval* and *Text/Document Processing* deal with data which is unstructured, a primary approach is to build models based on term/document indexes. This approach has been found to be noisy, due to the problems of polysemy and synonyms. It is computationally expensive as well, due to the large number of terms which can exist in data [12]. One approach to mitigating this problem is found in LSA / LSI [12]. LSA utilizes singular value decomposition (SVD) on a matrix of terms by documents, to discover useful *artificial semantic concepts* based on the co-occurrence of terms in documents. These *concepts* can be subsequently used to replace the individual terms as descriptors of the documents, i.e indexing based on the discovered concepts (LSI). LSA/LSI has been shown to improve document retrieval by 1) mitigating the problems of polysemy and synonyms and 2) reducing the dimensionality of the problem [12]. Since its initial application in document retrieval, LSA/LSI has found use in several other *Information Retrieval* and *Text/ Document Processing* tasks.

The *concept* discovery carried out by LSA in information retrieval is very similar to message type extraction. Message types represent terms which frequently occur together in the events in an event log. Importantly, they represent *real semantic concepts* in the mind of the programmer who wrote the code, so it makes sense to utilize them when analyzing log files. Previous work has shown that this approach is promising [83, 2].

In this chapter, research which demonstrates three practical applications of message types based on IPLoM is presented. These applications all have practical roles to play in the alert detection framework proposed in this thesis.

Firstly, for humans, visualization has always proved invaluable in aiding the analysis of large amounts of data. Hence log analysis can benefit from the use of visualization. An interactive tool for visualizing event logs using treemaps [65] is proposed. Secondly, the efficient storage and retrieval of log events is important. By using message types as a means of imposing structure on event log messages, a structured schema for the efficient storage and retrieval of messages is proposed. Thirdly, using the entropy-based nodeinfo [55] alert detection mechanism as a case study, the benefits of using message type indexing (MTI) as an alternative to the term-based indexing approach of nodeinfo is demonstrated. The use of MTI was able to reduce the computational effort and memory requirements of nodeinfo by a hundred orders of magnitude without a drop in its detection capability.

The material presented in this chapter can be found in these publications [41, 46, 44, 42, 49].

4.1 Visualization

While log files are invaluable to a network administrator, the vast amount of data they contain can overwhelm a human and can hinder rather than facilitate the tasks of an administrator. Visualization is an effective means of aiding humans to make sense of large amounts of data and could prove useful with event logs. Such can provide summaries of the contents of an event log file to an administrator. Thus, speeding up the data analysis is needed during downtimes/security breaches and for the daily monitoring of the system.

For such a visualization to be effective, it would require an effective means of abstracting and clustering the contents of the event log. Message types can prove useful in this respect. This section describes a prototype visualization tool called LogView. It uses message types to impose a hierarchy on the contents of the log and visualizes them using treemaps [65]. A cardinal goal in the design of LogView was to provide a visualization which was interactive and dynamic.

The prototype was built using the prefuse visualization toolkit [66]. Visualization toolkits are becoming an increasingly popular means of creating information visualization tools as they help to reduce the time and effort required to build them. Other examples of such toolkits include Piccolo [54] and InfoVis [14].

The prefuse visualization toolkit is a software framework written using the Java2D graphics library. Its design is geared toward the creation of visualization tools which are dynamic and interactive [19]. The design of the prefuse toolkit is based on the information visualization reference model outlined in the work of Chi [7]. By providing reusable building blocks, for the building of custom visualization tools, prefuse goes beyond what is offered in other visualization toolkits.

4.1.1 Methodology

A treemap is a methodology for the visualization of hierarchical data which uses a 2-dimensional space filling approach. Each node occupies a rectangular area within the confines of the 2-d space; the size of occupied area is often proportional to some attribute of the node, e.g. size [65]. Treemaps were first proposed by Professor Ben Shneiderman of the University of Maryland in 1992. His aim was to produce a visualization of the directory structure of the file system of his 80MB hard disk with a technique which utilizes a space-constrained layout. Treemaps provide an alternative to the traditional node-link structure diagrams used for visualizing hierarchical data which require a large amount of space for large hierarchies. Hence, treemaps are particularly useful when visualizing large amounts of hierarchical data, as they allow data to be viewed in a confined space as well as providing an interface which can be useful for encoding other pieces of information, a convenience which is not readily available or convenient with node-link representations. This makes treemaps an excellent choice for visualizing log files.

A classic treemap is produced using the slice-and-dice treemap layout algorithm. This algorithm works by taking any node and a rectangular space which is to be filled with a representation of the hierarchy starting at the node. Then it partitions this space into a number of regions equal to the number of outgoing edges from the node. The size of each partition is proportional to the attribute value of the node at the end of each edge. If this process is started with the root node in a hierarchy, the algorithm proceeds to call itself recursively on each child node using the partition allocated to the child node as the rectangular space. To differentiate between partitioning at alternate levels in the hierarchy, spaces are partitioned vertically (slice) at even levels and horizontally (dice) at odd levels.

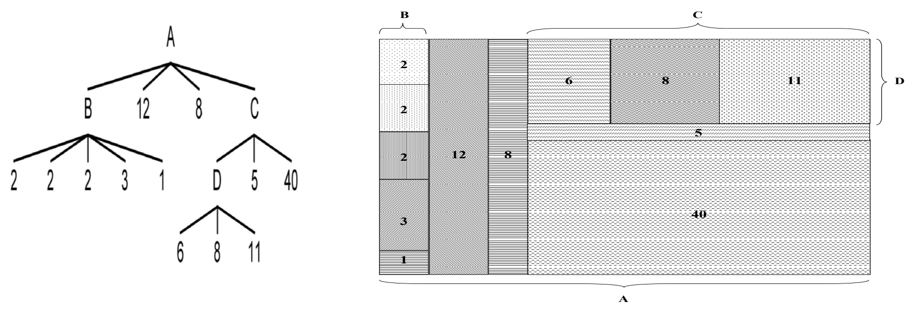
An example of how a treemap can be used to represent the hierarchical structure of a node-link diagram is provided in Fig. 4.1. Leaf nodes are represented by numbers in Fig. 4.1 (b) while internal nodes are represented with letters. The numerical value assigned to each leaf node is an indication of the size of the node. It is immediately apparent that these numbers could have been omitted in the treemap representation, as the size of each node is encoded by the size of its corresponding block in the treemap.

However, other treemap layout algorithms have been proposed to solve the problems associated with layouts created with the traditional slice-and-dice algorithm [4]. Some of these problems include low aspect-ratio, layout instability, order preservation in the face of dynamically changing data and the need to create layouts which are easy to search visually. Some of these layout styles include: Squarified, Strip, Cluster, Pivot-by-spilt, Pivot-by-size and Pivot-by-Middle. LogView utilizes the squarified treemap layout algorithm. Unlike the slice-and-dice algorithm, the squarified treemap layout attempts to ensure that each rectangular space is partitioned into units which are all roughly squares. By doing this, it maintains a low aspect-ratio even with hierarchies with a large number of nodes and large depth. Visualizing event logs as a hierarchy of nodes requires a large number of nodes. Hence the squarified layout is suitable for this task.

The data used in this project was collected on a Linux-based test server. The log files were collected on a per service basis and log files from four services were used. The services chosen include IMAP, POP3, SSH and HTTP. These services are explained in more detail below, while summary statistics of the event logs contents are summarized in Table 4.1 .

- **IMAP.** The IMAP service is an application layer protocol which allows a client to access his/her e-mails stored on a remote server over a TCP/IP connection.

- **POP3.** Like IMAP, POP3 is an application layer protocol which allows a client to access his/her e-mails on a remote server over a TCP/IP connection. It differs



(a) Tree Structure (b) Treemap representation

Figure 4.1: Treemap visualization of hierarchical data using the slice and dice algorithm.

from IMAP in that it does not allow persistent connections. This means that connections last for only as long as it takes to download messages, while IMAP allows clients to stay connected for as long as they require.

- **SSH.** SSH is an application layer protocol which allows two computers to exchange information over a secure encrypted channel. The information exchanged or transferred can be shell commands or files.

- **HTTP.** Probably the most popular application layer protocol in use today, HTTP is the protocol used for communication over the World Wide Web (WWW) and internal Intranets.

Table 4.1: Dataset Summary

Service	No. of Events	Period Covered
HTTPD	574,664	> 6 months
IMAPD	85,721	> 6 months
POP3D	86,190	> 6 months
SSHD	146,092	> 6 months

It is possible to cluster and visualize the log files for virtually all services. These four services were selected only as samples for the demonstration of LogView.

The data had to be processed into a suitable form before it could be used as input to the visualization tool. The three steps required to process the data are listed below.

- **Message Type Extraction.** Only relevant and meaningful message types were selected for use in the visualization. Descriptions of the message types which were selected for visualization from each service are outlined in Table 4.2. The name of each message type was assigned intuitively.

- **Event parsing.** This step involved the conversion of the message type descriptions into regular expressions. These regular expressions were used to parse the log files and separate the contents of the logs based on the message type to which they belonged; an event which did not belong to any message type was classified as an "outlier".

- **TreeML Conversion.** The visualization tool expects data in the TreeML format. The TreeML format is an XML-based file format designed for the purpose of specifying the data in a tree hierarchy. An outline of the TreeML format is given in Fig. 4.2. The final step was to convert the input into the TreeML file. The TreeML format requires that a number of data attributes be declared which can be assigned to the entities in the tree. Table 4.3 gives an outline of the data attributes used in the final TreeML files produced.

```
<tree>
 <declarations>
   <attributeDecl name="attr1" type="String"/>
   <attributeDecl name="attr2" type="Real"/>
    .
    .
    .
  </declarations>
  <branch>
      <attribute name="name" value="sample things"/>
          .
          .
        <branch>
            <attribute name="name" value="plants"/>
               .
               <leaf>
                    <attribute name="name" value="oak"/>
                     .
               </leaf>
                 .
                 .

       </branch>
       .
       .

 </branch>
</tree>
```

Figure 4.2: General outline of a TreeML file

Table 4.2: Message Type Summary

Service	Message Type	Description
SSH	Invalid User	Shell login request from a non registered user.
	Failed Reverse Mapping	Failed attempt to use getaddrinfo() to resolve a hostname.
	Spoof IP	A request from an IP address which may be spoofed.
	PAM Authentication Failure: Legal User	PAM login failure from a registered user.
	PAM Authentication Failure: Illegal User	PAM failure from an unregistered user.
	Failed keyboard-interactive/pam	Failed keyboard interactive or PAM authentication.
	No identification String	Shell login request without login information.
IMAP	Connection	A client connection.
	Disconnection	A client disconnection.
POP3	Logout	A client disconnection.
	Connection	A client connection.
	Login Failed	A failed login attempt.
	checkmailpasswd: Login Failed	A failed login attempt associated with checkmailpasswd.
	checkmailpasswd: Connection	A client connection associated with checkmailpasswd.
	checkmailpasswd: Logout	A client disconnection associated with checkmailpasswd.
HTTP	PHP Undefined Index: Fail	Error associated with an undefined array index which generates a failure message.
	PHP Undefined Index: Warn	Error associated with an undefined array index which generates a warning message.
	PHP Undefined Index: Ok	Error associated with an undefined array index which generates an information message.
	PHP Undefined Offset: 0	Error associated with an undefined numeric array index of 0.
	PHP Undefined Offset: 1	Error associated with an undefined numeric array index of 1.
	GET Request 500	HTTP get request with a status code of 500.
	GET Request 400	HTTP get request with a status code of 400.
	PHP Division by Zero	Error caused by division by zero.

Table 4.3: Data Attributes Used in TreeML Files

Attribute Name	Description
Name	A string describing the node.
Entries	An integer which indicates the number of events which are part of the entire tree rooted at that node.
Cluster	A string which represents the cluster description produced by SLCT.
Severity	A string describing the severity category of the even type. The categories are OK, WARN and FAIL.
Service	A string which indicates the service which produced the log entry.
Msg	The actual log event entry string.
Server	The name or IP address of the server on which the data was collected.

4.1.2 Results

An overview of the LogView interface is shown in Figure 4.3. The example shows the visualization of the contents of the SSH service log. The components of the LogView window, as can be seen in Fig. 4.3, include a service selection combo-box at the top of the screen, a dynamic query slider to the mid-right and the visualization itself to the mid-left. There are also search and detail panes which occupy the bottom-right and bottom-left of the screen, respectively. The labels for each cluster are the names given to each of the message types.

A detailed look at the visualizations created for each service type is shown in Fig. 4.4. Using such views, a system administrator can profile a particular service on a network mentally over time. For example it can be seen that the IMAP and POP3 event logs are relatively less complex when compared to HTTP and SSH, since all their entries were able to fall into defined clusters without outliers. Specifically, the IMAP visualization is the least complex with only two clusters.

Looking also at the SSH visualization, it can be seen that the majority of the entries fall into one cluster which cluster represents invalid login attempts. Further investigation showed that this was due to a prevalence of brute force login attempts on this server. In this case, LogView provided a very nice visualization of such an attack attempt on the SSH server. For the HTTP visualization, it is noticed that

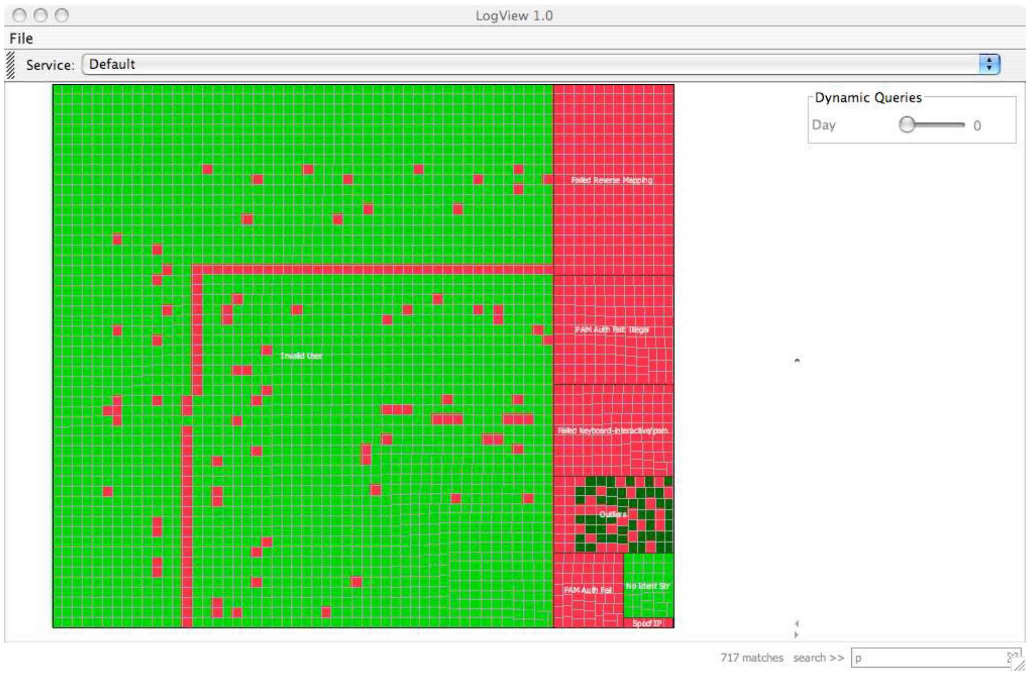


Figure 4.3: An Overview of the LogView Interface.

most of the clusters are PHP related. This is an immediate indication that this web server runs mostly PHP pages. Such a scenario would not occur on a web server which does not host PHP pages.

The data analysis and interaction capabilities of LogView are demonstrated using three possible tasks: searching, filtering and selection. A screenshot of a search over the SSH service showing log entries which contain the term “root” is shown in Fig. 4.5 (a); this is a task which an administrator might want to perform over this service to find the frequency of attempts to gain root access over SSH. In figure 4.5 (b), the filtering of the same visualization using the dynamic query can be seen. The filter is set to show only those entries which occur on the 27th day of the month; all other nodes are invisible. The view can be changed dynamically simply by dragging the slider. The slider has values in the range 0 - 31. The range 1 - 31 is for each of the possible days of a month and 0 (the default value) means “show all nodes”.

There might be times when the administrator has focused on a node of interest and wants more information on the node. LogView allows for such situations; by hovering a mouse pointer over the node, the actual log entry gets displayed in the detail pane below. An example of such a selection operation is shown in Fig. 4.5 (c).

For user convenience LogView provides the capability to zoom and pan the visualization. A screenshot of LogView zoomed out on the visualization of the SSH service is shown in Fig. 4.6 (a), while Fig. 4.6 (b) shows the tool zoomed in and panned to the left over the same visualization. Such utilities could prove valuable during the analysis of a log file.

4.1.3 Discussion

LogView demonstrates practically how message types can be incorporated into event log visualizations to make them more meaningful. The message types provide a means of viewing the flat structure of a event log as a hierarchy and by using treemaps, a visualization which makes effective use of space is made possible.

Due to its interactive nature, LogView is able to abstract the contents of the log file while still allowing the user to see the actual log file. As the size of event logs continue to increase in size, LogView provides a prototype for an effective tool for visualizing log files. It should be noted that while the hierarchy used in this prototype had only three levels, with message types as the middle tier, it is possible to encode other pieces of information at other levels in the hierarchy (e.g. the sources of the events). This will give rise to richer and more informative visualizations.

4.2 Message Type Transformation

In this section, methods for using message types to enforce structure on event messages are discussed. These methods are referred to as MTT, which provide more concise representations of event messages. They are important to the understanding of the application of message types to model building and indexing. In model building they can be used for dimensionality reduction, while using them in indexing can lead to space savings and of course faster and more intuitive searches.

The first MTT technique is called “Phrasal MTT”, see Fig. 4.7. It breaks up the message using the position of wild-card tokens in the message type descriptions as delimiters. Each group of constant tokens is treated as one term and replaced with a unique term throughout the log data. This MTT was designed to transform the message with minimal disruption to its original format. This technique will be called

MTT-1 for the rest of this chapter. The second MTT technique transforms a message by first representing it using a unique term which represents its message type. It then appends its variable tokens to the transformation in the same order they appeared in the original message. The encoding of token positions is very important in log analysis, hence the importance of maintaining the order of the variable tokens in the original message format. This method called, "MTT with variables," will be called MTT-2 for the rest of this chapter. The third MTT technique makes the most drastic changes to the data. It replaces a message with a token representing its message type and ignores its variables completely, see Fig. 4.9. The intuition here is that variable tokens may not be useful in certain kinds of log analysis, (e.g. the task of identifying alerts), the message types being more important. This MTT technique called "Full MTT" will be called MTT-3 for the rest of this chapter.

4.3 Storage and Retrieval

The material in this section demonstrates the use of message types as a means of imposing structure on event log messages for storage and retrieval using a structured schema. Not only would this improve the efficient storage and retrieval of log events, but it will ensure that the contents of event logs will be stored in a format which makes them ready to use in other log management and analysis tasks as well.

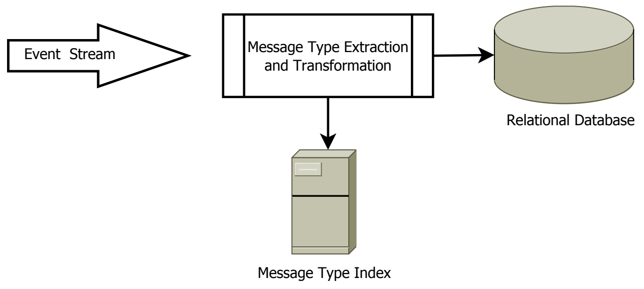
4.3.1 Methodology

MTT is the most important part of the methodology. With MTT, the unstructured message of an event can be transformed *losslessly* into an alternate format which is structured and removes redundancy. This new structured format can be exploited to provide an alternate schema for storing system log events.

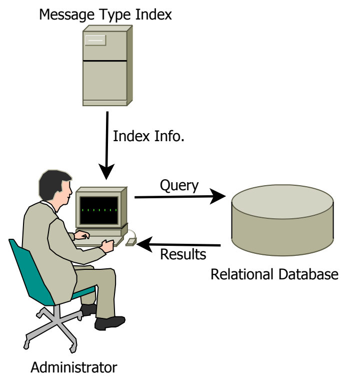
As it is intended for the transformation to be *lossless* and compact, the MTT-2 technique, Fig. 4.8, is proposed for this task. Unlike the other MTT techniques, the MTT-2 technique produces a more concise representation than MTT-1 and does not result in information loss as with MTT-3.

Most modern computer systems, especially those which follow the *syslog* format [38], would have a schema similar to the one shown in Fig. 3.1 in Chapter 3, i.e. {*Timestamp, Host, Class, Facility, Severity, Message*}. An alternative schema which breaks the events log based on message types and the creation of a message type index which maps to each event log is suggested. The schema for the message type index will be of the form {*MsgTypeID, MsgTypeFormat*}, while the schema for each message type log will be of the form {*Timestamp, Host, Class, Facility, Severity, Var1, Var2,....,VarN*}. The $Var1 \dots VarN$ fields will store each of the variable tokens in the message; as each one of these tables will contain only events of the same message type, the value of N will be constant for each table and the events stored therein. A visual depiction of how a stream of messages can be stored and retrieved is provided in Fig. 4.10.

With the advent of *Nested DBMS*, which differ largely from the more common relational DBMS in their ability to allow a schema which groups similar fields within a single field and their support for variable length fields, an alternate structured schema can be proposed. The message type index will remain the same while a single log will be maintained. Thus, the schema for the single log table using a *Nested DBMS* will be of the form {*Timestamp, Host, Class, Facility, Severity, MsgTypeID,* {*Var1, Var2,....,VarN*}}.



(a) Event Storage



(b) Event Retrieval

Figure 4.10: Proposed Event Storage and Retrieval Procedure.

This new storage and retrieval system based on the proposed schema aims to provide the dual benefit of: 1) Faster searches, and 2) Space savings on the disk. The space savings which will be achieved with MTT should not be confused with compression which is achieved with tools such as *zip* with which an *archived* document is created which cannot be used without *unzipping*. On the other hand, with MTT a *live* document that can be used as it stands is created. The use of MTT does not prevent the use of tools such as *zip* when an archived document needs to be created.

The datasets utilized to evaluate this component of the framework are four HPC datasets [56] which are publicly available in the USENIX Computer Failure Data Repository [75]. These datasets represent probably the largest set of publicly available HPC logs, covering approximately 111GB of data containing almost a billion events. The four datasets utilized are Blue Gene/L (BGL), Liberty, Spirit and Thunderbird (Tbird). Each line in the log data contains an actual event from the HPC machine which produced the log plus four additional fields, which are added to aid parsing. The four fields represent: the alert category (a "-" meaning no alert category), the Unix time-stamp, the date and the identifier for the device which generated the event. After these four fields comes the actual event as reported in the logs. The event consists of six fields: the first field represents the time-stamp, next comes the message source, the event type (the mechanism through which the event is reported), the event facility (the reporting component), the severity and the free form event description or message. For this work, the free form event description is of special interest, even though analysis is performed using other fields. For instance, the reporting device identifier enables the grouping of the events in any log based on the functional category of the device that produced the event. Using this information, the events in each dataset can be split into functional node categories. Thus, in the following evaluations, only the results for the *Compute* node category of each log are presented. This functional grouping represents over 73% of all events in each of the logs, hence it is safe to assume that results for this category should be representative of results over the entire dataset. The statistics of the HPC logs are given in Table 4.4.

Table 4.4: HPC Log Data Statistics

Log	**Days**	**Size(GB)**	**# Events**
Blue Gene/L	215	1.207	4,153,009
Liberty	315	22.820	200,940,735
Spirit	558	30.289	218,697,851
Thunderbird	244	27.367	155,403,254

To evaluate the space savings gained by transforming the message portions of events using MTT, message types are extracted from the logs and then the events are transformed using the extracted message types, creating a new log file in the process. The size of the transformed file and the original file are then compared to see how much space is saved, if any, by the transformation. To show that the proposed schema will lead to faster searches, the size of indexes which will be required to index the event log before and after transformation with MTT are compared. In both cases the percentage reduction, which is calculated using the formula in Eq. 4.1 below, is reported.

$$\%Reduction = 1 - \frac{Transformed\_Size}{Original\_Size} \tag{4.1}$$

4.3.2 Results

Indexing is a means of increasing the speed of searches of textual data. In the case of event logs, searches are mostly done through sequential searches of the lines of the event log using the *grep* command. To make this search any faster, an index of the unique tokens in the event log would have to be created. With the new schema proposed, the size of such an index can be reduced significantly. The results of the reduction in the number of terms required for an index are shown in Table 4.5. The average reduction in the number of terms is 99%, which implies a hundred order magnitude reduction in the time required for searches.

For example, should an administrator be interested in retrieving all messages of the form *"Connection from * port *"* or all messages with the word *"Connect"* from an event log stored in log.txt using the traditional schema, he/she could issue the following commands: *(grep "Connection from * port *" log.txt)* or *(grep Connection log.txt)*, respectively. In both cases, such a search would in the worst case require a sequential search through the entire event log. Even with an index, the search could take a significant amount of time since such an index would be very large (see Table 4.5). With the proposed schema a search for messages of the form *"Connection from * port *"* in the worst case would require a search through the message type index database for the message type ID linked to the messages of the form which the administrator seeks, leading straight to the required event log, which would be returned to the administrator (see Fig. 4.10). With the proposed schema a search for messages with the word *"Connect"* in them would require, firstly a search through the message type index database for the message type IDs of all message types which contain the word *"Connect"* and then secondly, a sequential search through the event logs of message types that do not contain the word *"Connect"*. In both cases, the proposed schema leads to faster search times with queries of the first type being significantly faster. Since message types are fundamental concepts in event logs, it is safe to assume that most searches issued by administrators would be of the first type. It is noted that the proposed schema provides the possibility of using *SQL* type queries when searching the event log.

Table 4.5: Percentage Reduction in # of terms

	Original	**Transformed**	**% Reduction**
	#Terms	**#Terms**	
BGL	491,768	399	99.92
LIberty	1,129,767	481	99.96
Spirit	898,111	854	99.90
Tbird	4,877,988	1,262	99.97
Avg.			99.94

One of the advantages derived from MTT is the removal of redundancy in the event logs. The results show an average reduction in size on disk of approximately 28%, with a best case scenario of approximately 38% reduction (see Table 4.6). This percentage reduction in file size can be significant in most cases, being measured in GBs, or even TBs on most modern computer systems. Indeed, the space gain from the transformation of the *Tbird* file is in the order of 4GBs. As this transformation is "lossless" and is not mutually exclusive to further compression using other techniques, these results show that MTT can provide significant space savings for the storage of yet-to-be-archived event logs.

4.4 Building Computational Models

Can computational models built using the contents of event logs benefit from the use of message types? That is the question which the material in this section attempts

Table 4.6: Percentage Reduction in File Size

	Original	**Transformed**	**% Reduction**
	Size(MB)	**Size(MB)**	
BGL	160	126	21.25
LIberty	8,563	5,271	38.44
Spirit	14,974	12,416	17.08
Tbird	12,583	8,007	36.37
Avg.			28.29

to answer. This is explored using the entropy-based alert detection mechanism of nodeinfo [55] as a case study. As originally defined, nodeinfo does not utilize message types for alert detection; it uses the tokens in the message in their raw form. This can make the building of alert detection models using nodeinfo expensive in terms of both computational and memory requirements. The previous section has shown that the number of terms in log files can be reduced by up to 99% by using MTT-3. If this new set of terms can be used with nodeinfo without a significant decrease in its detection capability, it would provide significant reductions in the computational and memory requirements of nodeinfo.

4.4.1 Methodology

Central to the nodeinfo framework is the concept of a *nodehour*. Given any event log E, a nodehour H_j^c can be defined as a spatio-temporal grouping of lines produced by a single node (c) within a one hour interval in tune with wall clock time [69]. In this case, H_j^c denotes the j^{th} nodehour for node c. For each line e_i in the event log, the reporting time and source node are determined by the timestamp (t_i) and reporting node (c_i) fields. Nodehours form the basis of the decomposition of an event log for analysis.

Nodeinfo bases its assessment of each nodehour on the information content of the individual tokens, t_1 to t_p in the free form message (m_i) field of an event e_i, with regard to its source. To incorporate the encoding of token positions into the framework, the concept of a *term* is introduced. A term is formed by concatenating each token with a number corresponding to its ordinal position in the message. For each token t_j in message m_i a *term* $w_j = t_j.j$ is created. Now, let W be the set of unique terms and let C be the count of nodes on the network. A $|W| \times C$ matrix $\mathbf{X}$ is computed, where $x_{w,c}$ is the count of the number of times term w appears in messages having node c as source. Then it is possible to use matrix $\mathbf{X}$ to compute vector $\mathbf{G}$ with cardinality $|W|$, where each element g_w of $\mathbf{G}$, is calculated using Eq. 4.2. The $p_{w,c}$ component of Eq. 4.2 is calculated using Eq. 4.3. $p_{w,c}$ is the probability that term w is produced by node c. The output of Eq. 4.2 corresponds to *1* plus each term's Shannon information entropy over the nodes of the network [55]. Its value ranges between *0* and *1*, with *0* signifying low information content for the term and *1* signifying the highest information content possible. Terms with high information content are more likely to indicate conditions which are of interest to an administrator and could be alerts.

The second step assigns a nodeinfo score to each nodehour based on the information content (measured by g_w) of the terms contained in the nodehour and how many times they appear. Let H be the set of all nodehours, a $|W| \times |H|$ matrix $\mathbf{Y}$ is defined, where $y^c_{w,j}$ is the count of the number of times term w appears in nodehour H^c_j. Then the nodeinfo score for nodehour H^c_j can be calculated using Eq. 4.4. The $NodeInfo(H^c_j)$ value computed by Eq. 4.4 represents the magnitude of the vector of counts of terms contained in Nodehour H^c_j weighted by the information content of the term. The information content is determined by each term's value in vector $\mathbf{G}$.

$$g_w = 1 + \frac{1}{\log_2(C)} \sum_{c=1}^{C} p_{w,c} \log_2(p_{w,c}) \tag{4.2}$$

$$p_{w,c} = \frac{x_{w,c}}{\sum_{c=1}^{C} x_{w,c}} \tag{4.3}$$

$$NodeInfo(H^c_j) = \sqrt{\sum_{w=1}^{|W|} (g_w \log_2(y^c_{w,j}))^2} \tag{4.4}$$

A ranking of nodehours based on their nodeinfo scores can be established. Nodehours with high nodeinfo scores are considered more likely to contain alerts than those which are lower in the ranking. For more details on the nodeinfo framework please see, [69, 55].

Experiments were performed to evaluate the effect of incorporating message types into the alert detection mechanism of Nodeinfo. Two evaluation methodologies were used. In the first evaluation, the nodeinfo framework was used exactly as described in [55]. This evaluation provides a baseline for the performance analysis. In the second experiment, the terms in the log were replaced using MTT-3 (see Fig. 4.11). In this case, the goal is to achieve equal or better performance while using the terms derived with MTT-3. Achieving this would suggest that the use of message types in building the alert detection model reduces the computational effort greatly. MTT-3 reduces the number of terms in the log about 99%. A similar reduction in computational effort can be claimed as the amount of computation is determined by the size of the matrices, **X** and **Y**.

The *binary scoring* metric as defined in [55], which defines the TP , FP, TN and FN, is utilized for measuring the performance. With these values, Precision and Recall values are determined as defined by Eqs. 3.3 and 3.4 in Chapter 3. To produce the Precision-Recall graphs, which are presented in Figs. 6.10, 4.13, 6.12 and 4.15, R_k is defined as the set of nodehours formed by taking the top k nodehours in a list of nodehours sorted using their nodeinfo scores at the end of an experiment. The value of k is varied from minimum to maximum and Precision and Recall values are calculated for each value of k. This set of Precision and Recall pairs are used in generating the Precision-Recall plots.

The datasets used in the experiments are the same datasets listed in Table 4.7. These datasets are appropriate for the following reasons.

- The events in these datasets have been have been labelled previously as alerts and non-alerts by domain experts. This determines the ground truth with which to compare the results of automatic analysis.

- The fact that the datasets are all publicly available adds to reproducibility of the results presented.

- Previous work in the development and evaluation of Nodeinfo has used these datasets [69, 55].

Details of the hardware architecture, configuration, characteristics, log collection methods and alert identification policies of these datasets and the systems which produced them can be found in [56].

The Nodeinfo framework relies on the assumption that *"Similar computers correctly executing similar work should produce similar logs"* [69]. For this reason log events from similar nodes need to be analyzed together for the framework to work effectively. To this end, the messages in the test datasets were separated based on the functional roles of the nodes which produced them, leading to thirteen categories: four categories for BGL, i.e. *Compute, IO, Link and Other*; and three categories each for Liberty, Spirit and Tbird, i.e. *Compute, Admin and Other*. A fourth category for the Tbird dataset (*SM*) was not included in this analysis as it had no identified alerts. The four *Other* node categories are not functional groupings of messages but consist of all messages which could not be placed in any of the other categories or had unknown source information. The data statistics of the resultant datasets based on functional groupings are detailed in Table 4.7.

Table 4.7: Functional Group Data Statistics

	# Events	**# Nodes**	**# Nodehours**	**% Alerts**
BGL-Compute	4,153,009	32,770	1,581,845	4.42
BGL-IO	400,923	1,024	219,722	38.22
BGL-Link	2,935	517	1,395	2.37
BGL-Other	191,096	2,167	13,666	0.43
Liberty-Compute	200,940,735	236	1,748,865	0.29
Liberty-Admin	52,211,676	2	27,162	0.04
Liberty-Other	12,416,820	6	44,447	0.22
Spirit-Compute	218,697,851	512	6,648,719	0.19
Spirit-Admin	41,847,257	2	26,216	3.10
Spirit-Other	11,753,861	7	57,532	0.25
Tbird-Compute	155,403,254	4,514	14.520,204	0.17
Tbird-Admin	15,306,749	20	100,740	0.02
Tbird-SM	19,109,810	2	8,859	0.00
Tbird-Other	21,392,379	1,319	626,030	0.02

4.4.2 Result

The results of running the original Nodeinfo framework on the original and transformed data is provided in Figs. 6.10, 4.13, 6.12 and 4.15. While the results differ for each set of nodes, the overall conclusion from the results is that it is possible to achieve a similar if not better result by modifying the data using the MTT-3 terms.

Similar results were achieved while modifying the data using MTT-1 and MTT-2 [44]. However, as these MTT techniques did not provide a significant reduction in the terms, their use is not justified as they do not reduce the computational effort significantly.

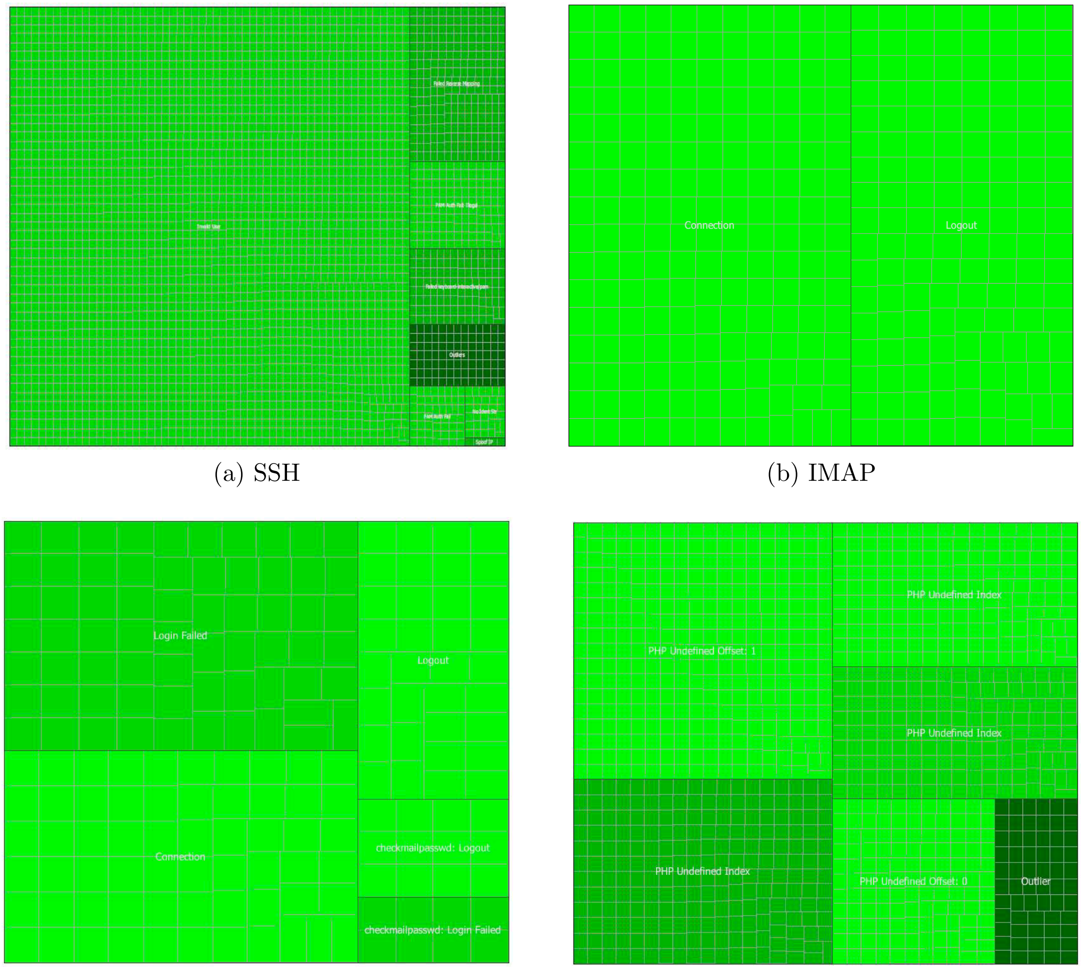
By introducing MTI after MTT a ~99% reduction in the size of the term vector utilized in the nodeinfo framework can be achieved, leading to an equal reduction in computational effort.

Apart from showing the improvements which the proposed changes can provide for alert detection in system logs, this work demonstrates practically the advantage which can be achieved by building models of event log data based on the message types they contain. With the introduction of algorithms like IPLoM, accurate automatic extraction of message type descriptions from log data has become a possibility.

4.5 Discussion

Message types form natural semantic concepts in log files. For this reason they should be used when free form messages are required for log file analysis. Unfortunately, the fact that message types are not always known apriori has hindered their use in system log analysis. The advent of several automatic message type extraction schemes has allowed message types to be incorporated into event log management and analysis mechanisms. Just as LSA has found use in text processing beyond the task of document retrieval, which it was initially designed for, it is likely that message type analysis of log files can find use in automatic log analysis beyond the examples mentioned in this chapter.

Since it is possible to accurately extract message types automatically, the methods described in this chapter can find practical implementation in any system, as the process can be automated fully. While all the application examples are important to the alert detection framework proposed in the thesis, the rest of the thesis will focus primarily on the alert detection component. Building on the success of the incorporation of MTI into nodeinfo, a novel alert detection mechanism, i.e. STAD, will be developed.



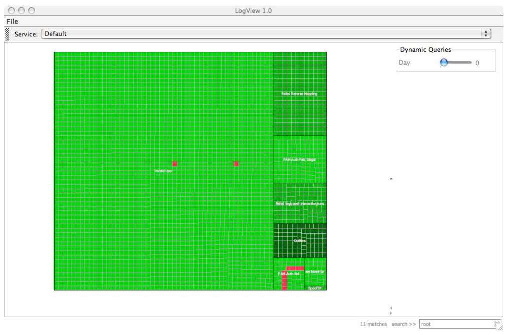
(a) SSH

(b) IMAP

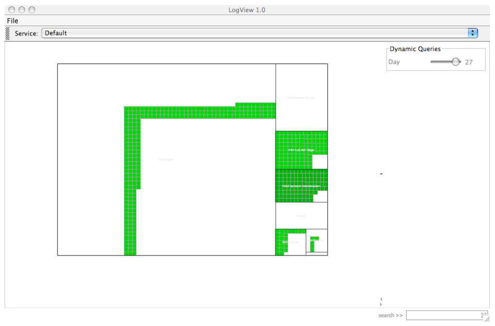
(c) POP3

(d) HTTP

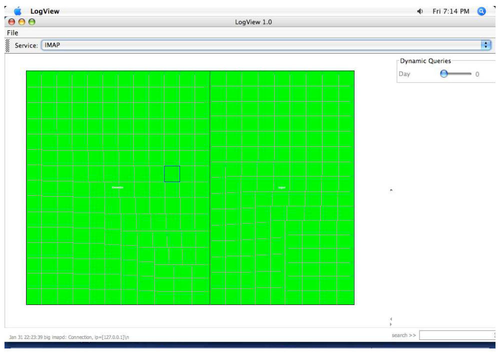
Figure 4.4: Figure showing the treemaps produced by LogView. (a)SSH (b) IMAP (c) POP3 (d)HTTP.



(a) Search showing all log events containing the term “root”



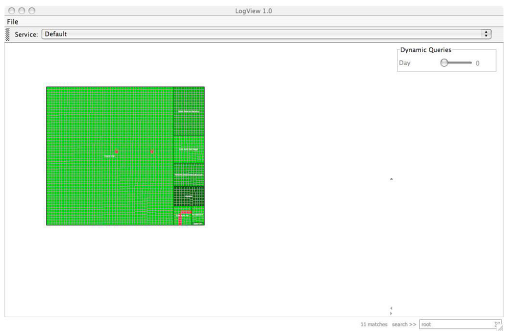
(b) Filtering the view to show only log events which occurred on the 27th day of the month



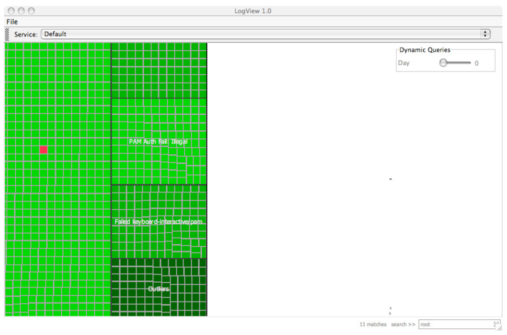
```
Jan 31 22:23:39 big imapd: Connection, ip=[127.0.0.1]\n
```

(c) Selecting a node causes the message of the log entry to be displayed in the textbox below the treemap.

Figure 4.5: Analysis Using LogView with the SSH service (a) Search (b) Filtering (c) Selection.



(a) ZoomOut



(b) ZoomIn and Pan-Left

Figure 4.6: Zooming and Panning with the SSH service.(a) Zoomout (b) Zoomin and Pan-Left.

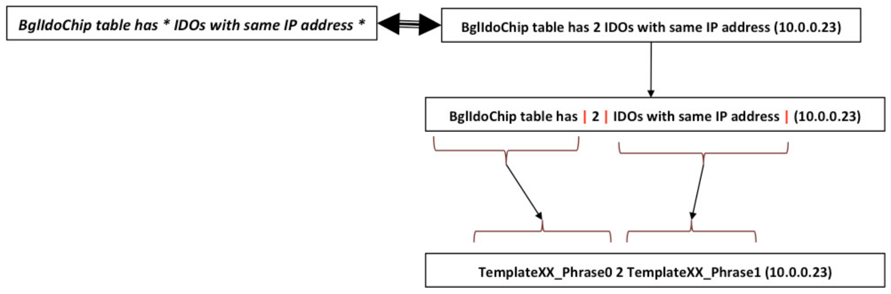

BglIdoChip table has * IDOs with same IP address *
BglIdoChip table has 2 IDOs with same IP address (10.0.0.23)
BglIdoChip table has | 2 | IDOs with same IP address | (10.0.0.23)
TemplateXX_Phrase0 2 TemplateXX_Phrase1 (10.0.0.23)

Figure 4.7: **Phrasal Message Type Transformation:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template which matches the message in the first box on the right. Each phrase consisting of constant tokens is replaced with a unique token. In the final box on the right, *XX* represents an ordinal number assigned to the message type and hence will change for different message types.

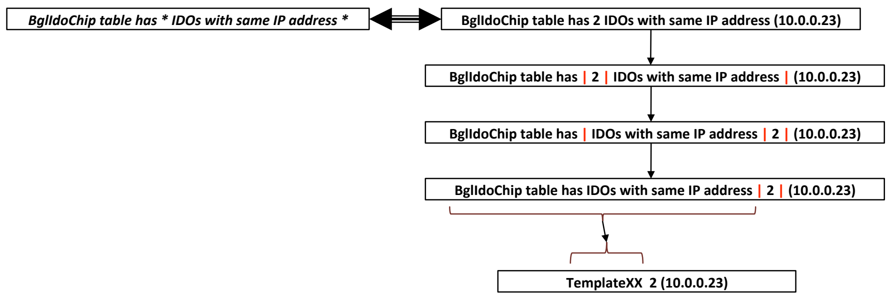

BglIdoChip table has * IDOs with same IP address *
BglIdoChip table has 2 IDOs with same IP address (10.0.0.23)
BglIdoChip table has | 2 | IDOs with same IP address | (10.0.0.23)
BglIdoChip table has | IDOs with same IP address | 2 | (10.0.0.23)
BglIdoChip table has IDOs with same IP address | 2 | (10.0.0.23)
TemplateXX 2 (10.0.0.23)

Figure 4.8: **Message Type Transformation with variables:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template which matches the message in the first box on the right. In the final box on the right, *XX* represents an ordinal number assigned to the message type and hence will change for different message types.

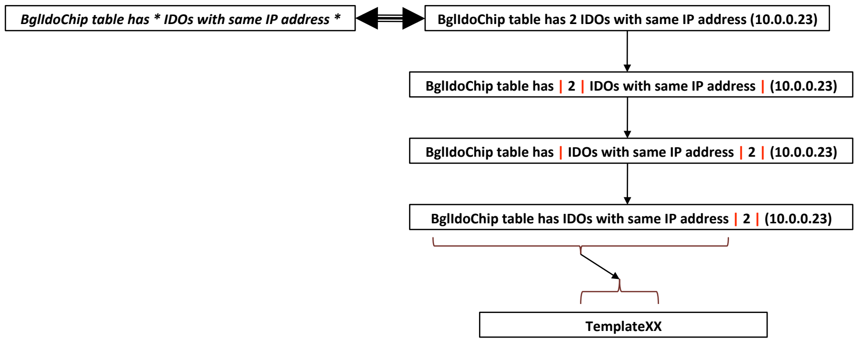

BglIdoChip table has * IDOs with same IP address *
BglIdoChip table has 2 IDOs with same IP address (10.0.0.23)
BglIdoChip table has | 2 | IDOs with same IP address | (10.0.0.23)
BglIdoChip table has | IDOs with same IP address | 2 | (10.0.0.23)
BglIdoChip table has IDOs with same IP address | 2 | (10.0.0.23)
TemplateXX

Figure 4.9: **Full Message Type Transformation:** The procedure starts with an individual message as contained in the first box on the right. The box on the left contains the message type template which matches the message in the first box on the right. This method is identical to the message type transformation with variables except that the variable values are discarded in the final transformation. In the final box on the right, *XX* represents an ordinal number assigned to the message type and hence will change for different message types.

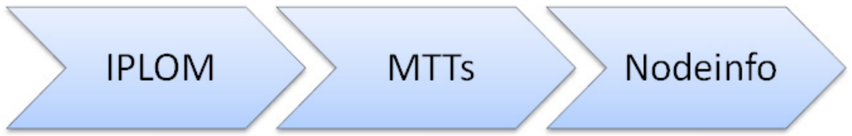
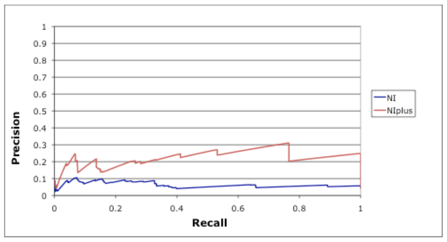
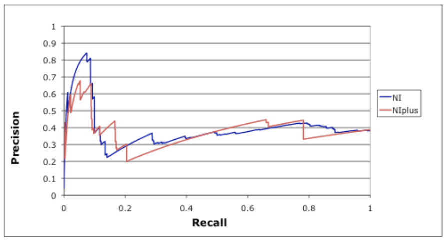


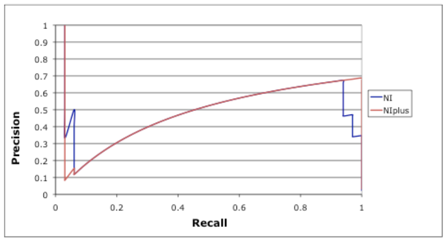
Figure 4.11: Process flow for experiments. The baseline for performance had the raw data input into Nodeinfo without preprocessing using IPLOM and MTTs.



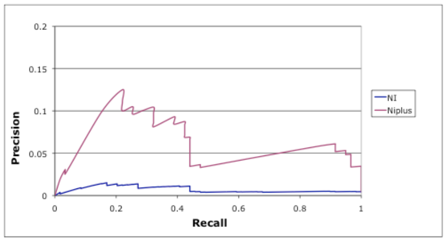
(a) BGL-Compute



(b) BGL-IO

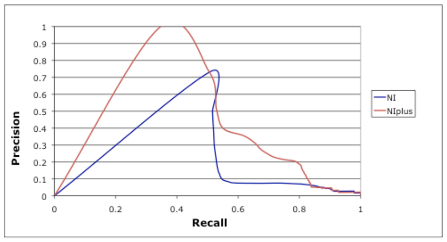


(c) BGL-Link

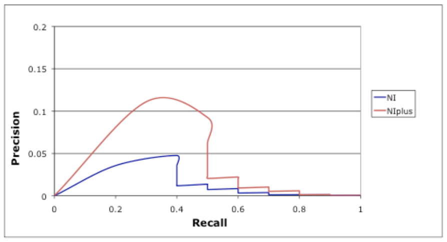


(d) BGL-Other

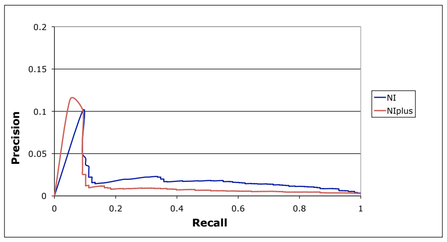
Figure 4.12: Precision-Recall plots for the BGL node categories using the original Nodeinfo framework (NI) and modified framework (NIPlus) obtained by transforming the raw log data using MTT-3.



(a) Liberty-Compute

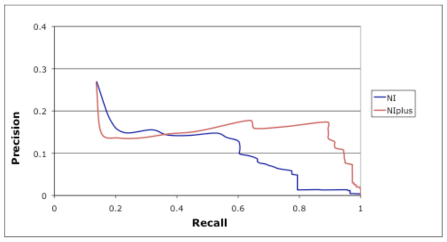


(b) Liberty-Admin

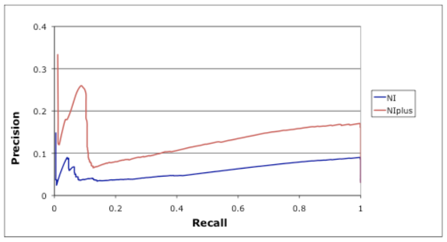


(c) Liberty-Other

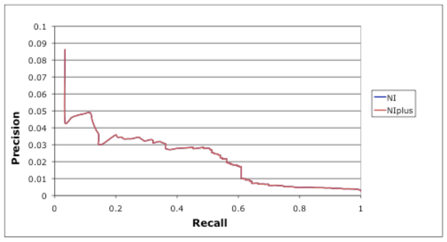
Figure 4.13: Precision-Recall plots for the Liberty node categories using the original Nodeinfo framework (NI) and modified framework (NIPlus) obtained by transforming the raw log data using MTT-3.



(a) Spirit-Compute

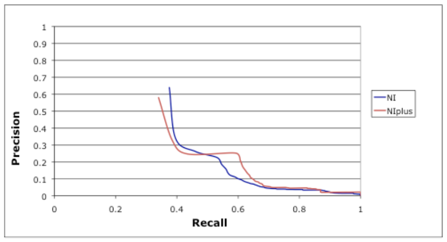


(b) Spirit-Admin

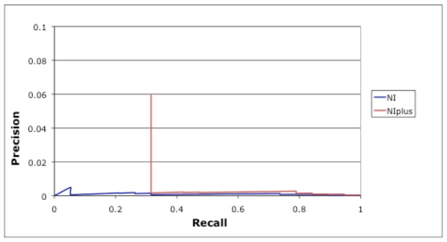


(c) Spirit-Other

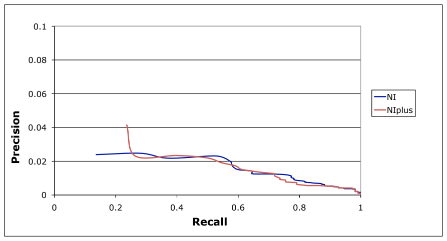
Figure 4.14: Precision-Recall plots for the Spirit node categories using the original Nodeinfo framework (NI) and modified framework (NIPlus) obtained by transforming the raw log data using MTT-3.



(a) Tbird-Compute



(b) Tbird-Admin



(c) Tbird-Other

Figure 4.15: Precision-Recall plots for the Tbird node categories using the original Nodeinfo framework (NI) and modified framework (NIPlus) obtained by transforming the raw log data using MTT-3.

Chapter 5

Spatio-Temporal Decomposition, Correlated Message Types and System State

Self-awareness is an important attribute for any self-managing system to have. A system needs to have a continuous stream of real-time data to analyze to be aware of its internal state. To this end, previous approaches have utilized system performance metrics and system log data to characterize a system's internal state. In order to utilize system logs for this task, the computation of strongly correlated message types is necessary.

System log analysis follows, encompasses or requires one or more of the following steps.

- **Unstructured Message Analysis :** Events in system logs are not homogenous entities: they contain structured and unstructured information. The unstructured information, namely free-form messages, pose a stumbling block to the automatic analysis of log data. Therefore, analysis of unstructured messages is required for further understanding of system logs. Message type extraction deals with this problem.

- **Indexing/Feature Creation :** This involves the creation of indexable features from the unstructured data in the form of message type *IDs* and message variables. MTT deals with this problem.

- **Event Correlation:** In most cases, it is unlikely that a single event in a system log can characterize system behavior. As such, it is important to find message types which are correlated in the system logs. Correlated messages are usually better indicators of system state. The event correlation problem has yet to be tackled in this thesis. In this chapter, it is shown that strongly correlated message types can be discovered without much computation.

The proposed method explores a natural behaviour of system logs when system log data is partitioned using source and time information. Such a system log partition will contain log data from a single source on the network over a unit period of time. This work highlights the observation that such partitioning of system logs leads naturally to partitions, which contain correlated message types, a previously unknown property of system logs. It demonstrates how the groups of partitions which contain correlated message types can be found by clustering the partitions based on their entropy-based information content. The *conceptual clusters* formed by the method are evaluated using cluster cohesion, cluster separation and cluster conceptual purity as metrics. *Conceptual clusters* are clusters in which objects in a cluster can be described by a concept, not just based on their distance from each other [18]. The results demonstrate that the proposed method produces not only well-formed clusters but also clusters which can be mapped to different alert states with a high degree of confidence, with regard to the different alert types identified by system administrators in the log data. Another advantage of this clustering technique is the fact that users do not need to specify the number of clusters before clustering begins.

The material presented in this chapter can be found in these publications [44, 47, 48, 50].

5.1 Related Work

The literature abounds with previous attempts at the automatic discovery of correlated messages which indicate system state in log data. One thing which is common among them is that for the most part they attempt to discover the correlated messages by analyzing the log temporally. The method introduced here differs because it introduces not only a *spatial* component to the analysis but also in that it requires very little computation to execute. Some examples of related work are given below.

The Loghound log data mining tool, which is an implementation of a frequent itemset mining algorithm, can be used for discovering correlated message types [77]. Loghound does this by decomposing the log using a fixed length time interval provided by the user. These log time intervals are viewed as transactions, with the message types reported in the interval as items. Using its frequent itemset mining mechanism, Loghound finds the frequent itemsets in these transactions. These frequent itemsets are viewed as the correlated messages in the log. In [34], Liang et al. propose a 3-step filtering algorithm for filtering failure logs from a high performance cluster (a BlueGene/L prototype), which compresses and categorizes the events in the log to understand failure behavior better. The filtering is carried out temporally.

In [33], the authors discover temporal relationships between message categories which are discovered using a modified Naive Bayes algorithm. The message categories are based on previously defined categories associated with the IBM CBE (Common Base Event) format [74]. These temporal relationships expose the temporally correlated message categories in the log. The authors propose that these relationships be visualized to monitor system behavior.

In [57], the authors assume the existence of known message types and use a process they refer to as *event summarization* to mine and rank temporal dependencies between event types. Temporal dependencies are mined using time series analysis and are ranked using a *forward entropy* technique [9]. These dependencies are visualized using an Event Relationship Network (ERN) [72, 58]. The ERN is used to interpret system behavior and derive rules for system management. In [36], Lim et al. utilize message de-parameterization to create message types from enterprise telephony system logs. Messages in the logs are replaced by these message types, which they refer to as *message codes*. The logs are then further analyzed using frequent itemset mining to discover correlated messages, which are useful in determining failure states in the system.

In [2], Aharon et al. propose the PARIS (Principal Atom Recognition in Sets) algorithm. This algorithm is able to detect *atoms*, i.e. sets of correlated message types which are produced as part of a normal process or failure activity. In this case, the message types have been mined previously. The authors propose the monitoring of these atoms through visualization as a means of detecting failure in system logs. Xu et. al proposed a PCA-based (Principal Component Analysis) framework for the detection of system problems through the analysis of console logs [83]. In their case, message types were extracted from source code, while correlated message types were discovered by tracking the variables reported in the message types. They argue that messages which report the same variable(s) are likely to be correlated.

5.2 Methodology

In this section, the intuition behind the proposed method of extending an entropy-based approach to system behavior characterization is described. During the process of evaluation of entropy-based alert detection, several approaches were tested on the HPC logs listed in Table 4.7 in Chapter 4. Specifically, three methods for assigning information content scores (ICS) to nodehours were tested. These methods will be referred to as *NI*, *NIUniq* and *NIMax*, which are represented by Eqs. 5.1, 5.2 and 5.3, respectively. In each of these equations H_j^c refers to the j^{th} nodehour of node c.

The details of Eq. 5.1 will not be discussed here, as this equation is identical to Eq. 4.4 already introduced in Chapter 4. Information content assignment using Eqs. 5.2 and 5.3 proceeds in very much the same way as that done with Eq. 5.1. Hence, the first step of assigning entropy-based ICSs for each individual *term* which appears in the log is identical (see Eqs. 4.3 and 4.2 in Chapter 4). However, in the case of *NIUniq* and *NIMax*, *terms* would refer strictly to the tokens produced through MTT-3, as used with *NIPlus* in Chapter 4

For the second step in which an ICS is assigned to each nodehour, Eqs. 5.2 and 5.3 are used. In these equations, matrix $\mathbf{Z}$ is a matrix in which each entry $z_{w,j}^c$ only records unique occurrences of terms in the event data, i.e. $z_{w,j}^c$ is *1* when term w appears in nodehour H_j^c and *0* otherwise. This is different from the matrix $\mathbf{Y}$ used in Eq. 5.1, in which $y_{w,j}^c$ is the count of the number of times term w appears in Nodehour H_j^c. Therefore *NIUniq* assigns an ICS to a nodehour based on the magnitude of the vector of information content values of the terms contained in nodehour H_j^c. While *NIMax* assigns an ICS to each nodehour H_j^c which is the highest entropy value assigned to any *term* which is reported during the nodehour.

$$NI(H_j^c) = \sqrt{\sum_{w=1}^{|W|}(g_w \log_2(y_{w,j}^c))^2} \tag{5.1}$$

$$NIUniq(H_j^c) = \sqrt{\sum_{w=1}^{|W|}(g_w * z_{w,j}^c)^2} \tag{5.2}$$

$$NIMax(H_j^c) = max_w(g_w * z_{w,c,j}) \tag{5.3}$$

Results highlighted in [44, 47] and Appendix B, have shown that *NIUniq* results in improved alert detection accuracy and reduced computation during the alert detection process. However, the research presented in this section is not concerned with accuracy in alert detection. This research is concerned with using ICSs to determine correlated message types, which in turn represent system states.

During evaluations, a strong clustering of nodehours around single ICS values was observed which was particularly pronounced when *NIUniq* (i.e. Eq. 5.2) was used. Such clustering of values could be considered odd, since ICSs are real numbers which can take any value in the range $[0, \infty)$. The graphs in Figs. 5.1, 5.2, 5.3 and 5.4 show the scatter plots of the ICSs for nodehours from the datasets listed in Table 4.7. In each graph, the y-axis represents the ICS for a nodehour using Eq. 5.2, while the x-axis represents each individual nodehour sorted according to its ICS. It can be seen that the clustering manifests itself in all the graphs, while being most pronounced with the BGL-Link category.

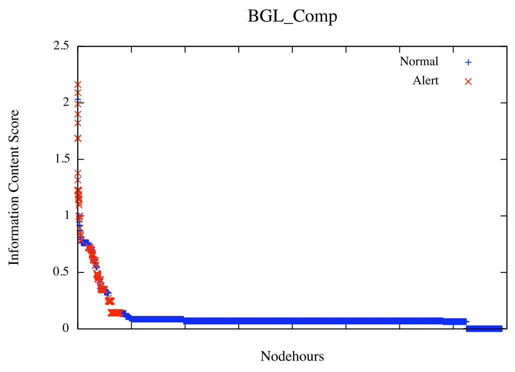
The following discussion tries to explain what may be responsible for this observation. Consider a set of distinct objects X which you wish to sample (with replacement) and distribute into a number of bins (each bin acting as a *bag*, which can contain several instances of the same object from X), with the following constraints:

1. If the number of bins is n, then $|X|$ should be $<< n$.

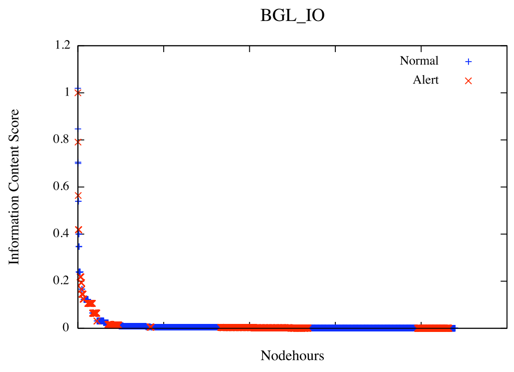
2. If Y_i is the set of unique objects in bin i, then $|Y_i|$ should be $<< |X|$ for most i.

If the sampling from X described above is carried out even with a random sampling method, it is easy to see how several bins could end up containing the same set of distinct objects. Constraint 2 reduces the number of possible distinct object combinations which can exist in any bin, while constraint 1 ensures that the chance for a combination to repeat itself is high. If the number of possible combinations of objects from X given constraint 2 is less than n, a combination repetition is guaranteed.

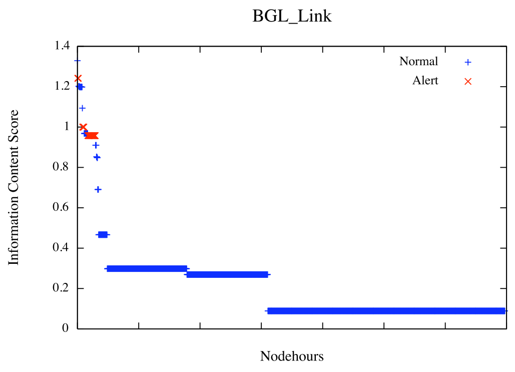
If X is assumed to be the set of message types which exist in a system log and the bins are assumed to be the nodehours in the system log, then the process described above can be reduced to the system log analysis domain with one major difference: the sampling from X will follow a Pareto distribution rather than a random distribution. Previous work [76, 77] has shown that the distribution of messages in system logs by



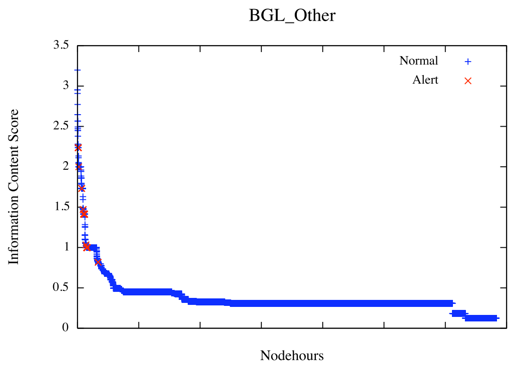
(a) BGL-Compute



(b) BGL-IO



(c) BGL-Link

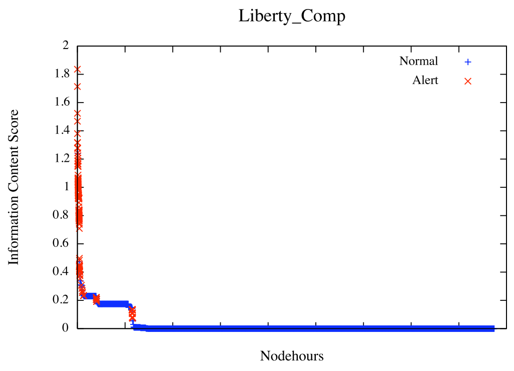


(d) BGL-Other

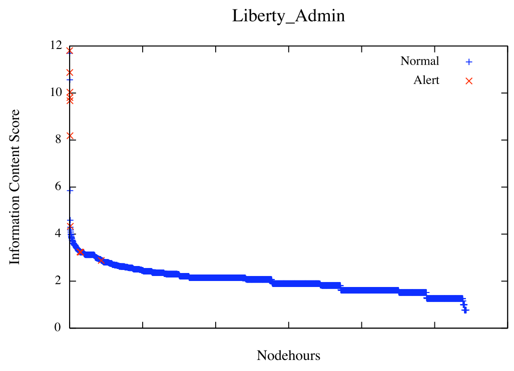
Figure 5.1: Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the BGL node functionality categories. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on ICS in the plot.

nature follow a Pareto distribution. This means that the sampling of objects from X will be biased in such a way that a small subset of the objects would be sampled more frequently than others. This biased sampling should accentuate the result of having several bins containing the same set of distinct objects.

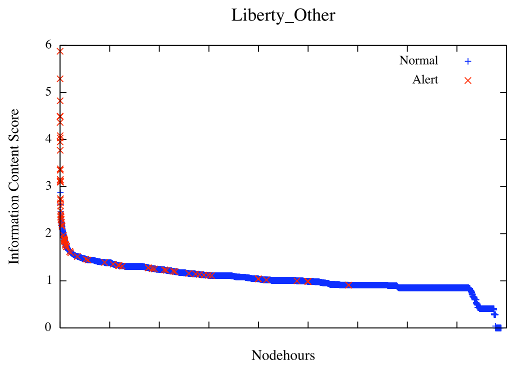
It is conjectured that the process described above is responsible for the strong clustering of nodehours around a single ICS as in Figs. 5.1, 5.2, 5.3 and 5.4. The ICS value derived from the use of Eq. 5.2, is in a way, a *hash* value for the set of unique message types in a nodehour, so a distinct ICS can be linked to one or more sets of unique message type combinations. This would explain why this observation is more pronounced when Eq. 5.2 is used to calculate ICSs for nodehours. Eq. 5.1 weights its results with the frequency of occurrence of each *term*, thus two nodehours can have



(a) Liberty-Compute



(b) Liberty-Admin

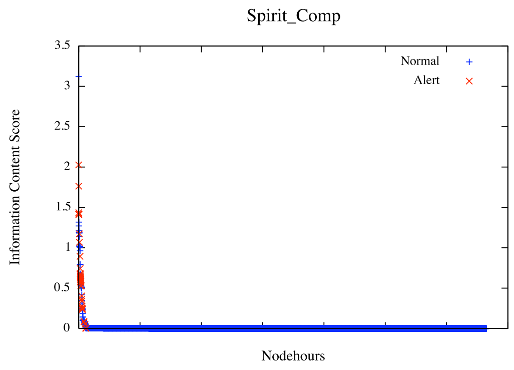


(c) Liberty-Other

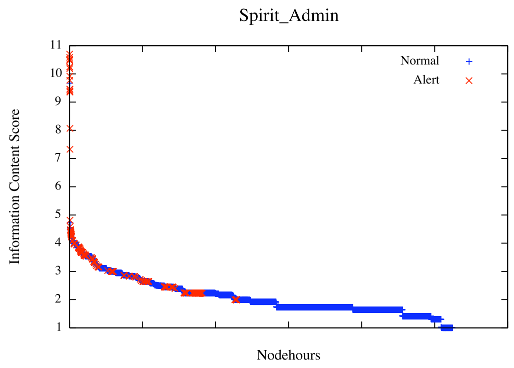
Figure 5.2: Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the Liberty node functionality categories. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on ICS in the plot.

the same ICS only if they have the same message type combination appearing at the same frequency. On the other hand, Eq. 5.3 does not take all the message types in a nodehour into consideration. This highlights another advantage of using Eq. 5.2 over the other methods evaluated. Previous work suggests that temporal filtering of system log messages could be beneficial for system log analysis [34] and Eq. 5.2, represents a form of implicit temporal filtering of system log messages.

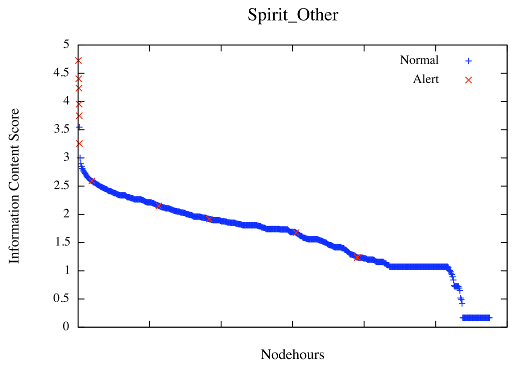
From Table 5.1 it can be seen that the constraints described earlier hold true for the datasets. The "# Msg-Types" column represents $|X|$, the "# Nodehours" column represents the number of bins n, while the "Msg-Types/Nodehour (Max)" and "Msg-Types/Nodehour (Avg.)" columns represent the maximum and average number of message types which can be found in each nodehour, respectively. Based



(a) Spirit-Compute



(b) Spirit-Admin



(c) Spirit-Other

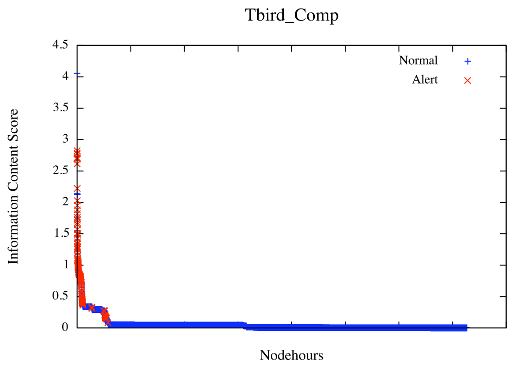
Figure 5.3: Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the Spirit node functionality categories. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on ICS in the plot.

on these observations, the following hypotheses are derived.

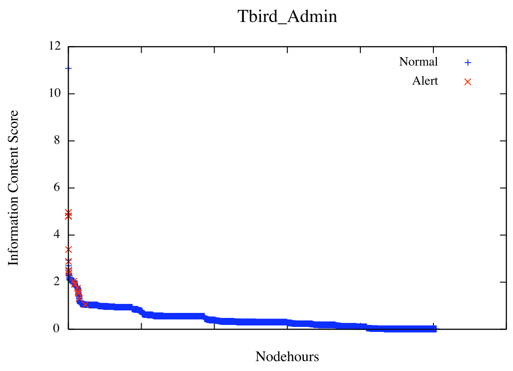
- Nodehours with the same ICS (based on Eq. 5.2), contain the same unique set of message types.

- ICSs, which occur frequently, represent nodehours which contain strongly correlated message types and represent some system behavior or characteristic.

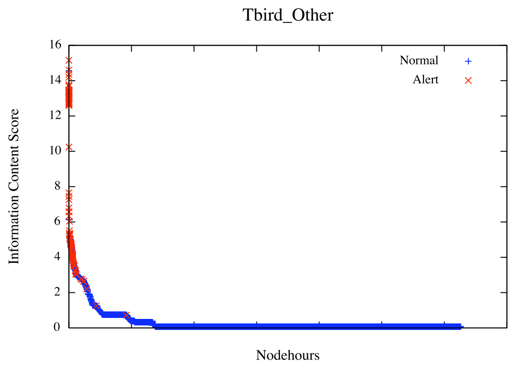
The work described in this chapter of this thesis attempts to validate these hypotheses. Should these hypotheses be validated, it can be concluded that the ICSs of spatio-temporal partitions of log data can serve as an effective means of discovering correlated message types and hence system state.



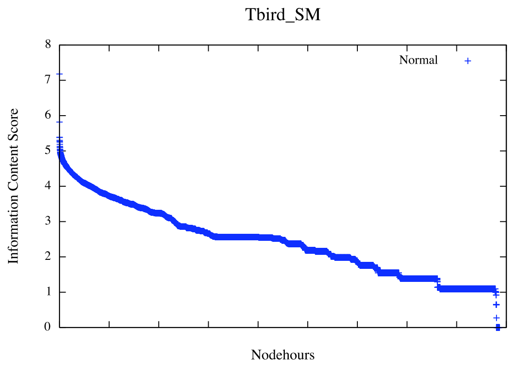
(a) Tbird-Compute



(b) Tbird-Admin



(c) Tbird-Other



(d) Tbird-SM

Figure 5.4: Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the Tbird node functionality categories. The plot differentiates between alert nodehours and normal nodehours; the Tbird-SM category has no alert nodehours. Nodehours are sorted based on ICS in the plot.

Table 5.1: Statistics of Message Type Distribution across Nodehours for HPC Data

	# Nodehours	# Msg-Types	Msg-Types/Nodehour (Max)	Msg-Types/Nodehour (Avg)
BGL-Compute	1,581,845	399	117	1.37
BGL-IO	219,722	49	5	1.11
BGL-Link	1,395	13	4	1.23
BGL-Other	13,666	97	16	2.71
Liberty-Compute	1,748,865	481	214	1.87
Liberty-Admin	27,162	601	195	5.9
Liberty-Other	44,447	510	166	11.89
Spirit-Compute	6,648,719	854	349	2.29
Spirit-Admin	26,216	443	151	6.79
Spirit-Other	57,532	707	228	10.93
Tbird-Compute	14.520,204	1,262	325	3.62
Tbird-Admin	100,740	627	179	4.69
Tbird-SM	8,859	597	254	12.54
Tbird-Other	626,030	1,387	356	2.13

5.2.1 Information Content Clustering (ICC)

The pseudo-code in Algorithm 5 describes a method developed as part of this thesis for the clustering of the nodehours based on their information content. Essentially, the algorithm creates each *cluster* as a bin which can be described using the tuple $(ICS, ``MaxEntropyMsgType'')$, where $ICS$ is an ICS value and $``MaxEntropyMsgType''$ is the *ID* of the message type with the maximum entropy value among all the message types which have instances in the nodehour. Intuitively, it would be sufficient to use only the ICS as a description for each bin. However, for the first hypothesis in Sec. 5.2 to be true, two nodehours with the same ICS should at least have the same message type with maximum entropy, given that this message type would be the highest contributor to the ICS in all probability. Hence, the addition of $MaxEntropyMsgType$ to the description. The addition of $MaxEntropyMsgType$ to the description has the effect of reducing collisions as well. All nodehours with the same values for the tuple will end up in the same cluster.

Algorithm 5 This pseudo-code describes the proposed method for clustering a set of nodehours.

```
Input: Set $S$ of nodehours with associated ICSs (Nodeinfo).
    Array $m_{max}$ which contains the message type with maximum entropy for each
    nodehour in $S$.
Output: Collection $S_1 \dots Sn$ of clusters of $S$.
1: for each nodehour $s$ in $S$ do
2:     Determine the cluster $S_i$ that $s$ belongs to as combination of its ICS and its
       maximum entropy message type.
3:     if $S_i$ exists then
4:         $s \in S_i$ {Add $s$ to $S_i$}
5:     else
6:         Define a cluster $S_i$ of $S$, such that $|S_i| = \emptyset$
7:         $s \in S_i$ {Add $s$ to $S_i$}
8:     end if
9: end for
   {Each cluster produced will represent the set of nodehours which have the same
   ICS and the same maximum entropy message type.}
```

A list of some of the properties which differentiate the proposed method from previous approaches is provided below:

1. **Spatial Decomposition :** Previous work has shown that one of the major mitigations against finding correlated messages in system logs is the fact that correlated messages may not always follow each other in sequence in the system logs [77]. The entropy-based information content approach of the proposed method requires that the system log be decomposed spatio-temporally both at the point at which entropy values are calculated for terms and the point where ICSs are assigned to nodehours. Nodehours are spatio-temporal partitions of the log data. This observation increases the chance which messages that follow each other temporally are likely to be correlated.

2. **Complexity :** The complexity of this approach is $O(n)$, where n is the number of spatio-temporal partitions, i.e. nodehours in this case.

3. **Interestingness :** Since each pattern belongs to a cluster which is partially defined by an ICS, a means to evaluate the *interestingness* of the pattern which has been found without further analysis is available. *Interestingness* refers to the likelihood that the pattern represents the occurrence of an event which would require the attention of an administrator.

4. **Pattern Length :** For the most part, the patterns do not contain sub-patterns of a larger pattern, thereby reducing the number of patterns found. This happens because the proposed method deals only with the entire message type combination found in a nodehour. Patterns mined from system log data could be just as large and complex as the log data itself, hence the production of only a small set of interpretable patterns is desirable [57].

5.2.2 Evaluations

In this section, the methods used in evaluating the quality of the clusters formed by the proposed technique are described. It should be noted that since the ICS associated with each nodehour is a real number, a precision level of 10 decimal places was set when testing the equality of the ICSs. This is important for the reproducibility of the results. Also, only clusters which had > 9 nodehours in them were considered, arguing that clusters with fewer nodehours might have resulted by mere coincidence. The experiments were carried out using all the datasets listed in Table 4.7.

The clusters formed at the end of the experiments were evaluated using three measurements, namely cluster cohesion, cluster separation and cluster conceptual purity. Before the means of assigning values to these metrics is described, two important *meta-concepts* are described (i.e. the distance between clusters and the cluster centroid).

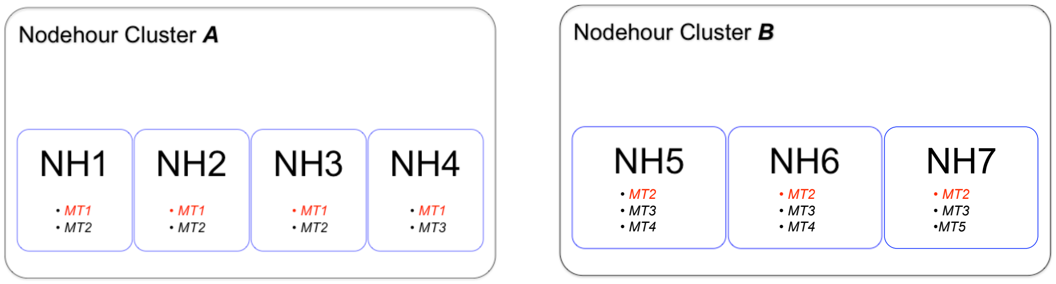
The cluster centroid for each *cluster* is an n-tuple which represents the message type combination which appears most frequently among the nodehours in the cluster. For example, in *Cluster A* in Fig. 5.5, the message type combination $(MT1, MT2)$ appears most frequently and therefore is the *centroid* for this *cluster*. Similarly the message type combination $(MT2, MT3, MT4)$ is the *centroid* for *Cluster B*.

The distance between two nodehours is defined using the standard function for finding the distance between objects defined by nominal variables [17], Eq. 5.4, where p represents the number of variables common to the objects and m represents the number of variables for which both objects match. For example, in *Cluster A* in Fig. 5.5, the message types common to $NH1$ and $NH4$ would be $(MT1, MT2, MT3)$, hence in this case $p = 3$ and the message types which match between them are $(MT1, MT2)$, and therefore, in this case $m = 2$. Thus, the distance between $NH1$ and $NH4$ would be 0.33. In this way, distances between pairs of nodehours and *cluster centroids* can be calculated.

$$d(i,j) = \frac{p-m}{p} \tag{5.4}$$

- **Cluster cohesion :** This measures the degree of similarity of the members of a cluster. It is desired that the members of a cluster be very similar. This is measured using the *Fwithin* statistic as defined in Eq. 5.5. It represents the standard deviation within a cluster, using the *centroid* as *mean*. In Eq. 5.5, μ_x represents the *centroid* of *cluster X*, while x_i represents the *ith* nodehour in *cluster X*.

- **Cluster separation :** This measures the degree of similarity between different



(a) Cluster A (b) Cluster B

Figure 5.5: Two example nodehour clusters formed using Algorithm 5. The enclosed boxes represent nodehours which belong to the cluster and the bulleted lists represent the set of unique message types which appear in the nodehour. In each bulleted list the message type highlighted in *red* represents the message type with maximum entropy which is common to all nodehours in a cluster based on Algorithm 5.

clusters. It is expected that clusters should be very dissimilar. This is measured using the *Distance* statistic as defined in Eq. 5.6. It represents the distance between two clusters X and Y using Eq. 5.4.

- **Conceptual Purity :** It is assumed that the proposed method produces *conceptual clusters*. It would be interesting to see whether the clusters formed could be linked to concepts and if so, with what level of confidence. *Conceptual Purity*, as used here, attempts to measure the degree to which the clusters formed meet this criterion. The datasets used in the experiments provide ground truth with respect to the *alert concepts* which exist in these datasets in the form of the alert categories assigned to each event in the log [56]. To this end, the conceptual purity of the *alert clusters*, i.e. clusters which contain a majority of alert nodehours, is measured. The process defined in Algorithm 6 determines the ratio of nodehours in an alert cluster which contain the signature for an alert category. It measures the degree to which the alert nodehours in a cluster can be linked to an alert category. A value of 1 for this ratio implies that all the alert nodehours in a cluster can be linked to the same alert category. Therefore, the cluster can be linked conceptually to the alert category with 100% confidence. For example, suppose that alert category β can be linked to $MT2$, i.e. $MT2$ is a signature for the error represented by β. In *Cluster A* in Fig. 5.5, $MT2$ appears in 3 out of 4 nodehours, hence the alert category to alert cluster ratio for *Cluster A* with respect to alert category β is $\frac{3}{4}$. This process can be repeated for all alert categories whose signatures appear in the nodehours of *Cluster A* and the same can be done for all the clusters. The average of these values represents the conceptual purity with respect to the alert categories defined in the system log.

$$Fwithin(X) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} d(\mu_x, x_i)} \tag{5.5}$$

$$Distance(X,Y) = d(\mu_x, \mu_y) \tag{5.6}$$

Algorithm 6 This pseudo-code describes the method for determining the degree to which the signature of an alert category can be linked to an alert cluster.

Input: Set M_{alert} of messages types which can be linked to alert category
Cluster S_{alert} whose conceptual purity with respect to the alert category we want to determine.

Output: Alert Category to Alert Cluster Ratio. [Range [0,1]]

```
1:  $ratio\_sum = 0$
2:  $count = 0$
3:  Determine the number $n$ of nodehours in $S_{alert}$
4:  for each message type $m$ in $M_{alert}$ do
5:    Determine the number $x$ of nodehours in $S_{alert}$ that contain $m$
6:    if $x > 0$ then
7:      $temp\_ratio = \frac{x}{n}$
8:      $ratio\_sum = ratio\_sum + temp\_ratio$
9:      $count++$
10:   end if
11: end for
12: $cluster\_ratio = \frac{ratio\_sum}{count}$
13: $Return(cluster\_ratio)$
```

5.3 Results

The results demonstrate that by using the proposed ICC method well formed and conceptually meaningful clusters can be produced. The statistics of the clusters formed are presented in Table 5.2. As mentioned earlier, only clusters of size > 9 are considered. The number of clusters formed with this restriction is given in the *"# Clusters (> 9)"* column and the total number of clusters formed irrespective of size is given in column *"Total Clusters"*. The *"% Nodehours"* column gives the percentage of nodehours which belong to clusters if only clusters of size > 9 are considered. The results demonstrate that, on average, ~92% of nodehours can be clustered using this approach.

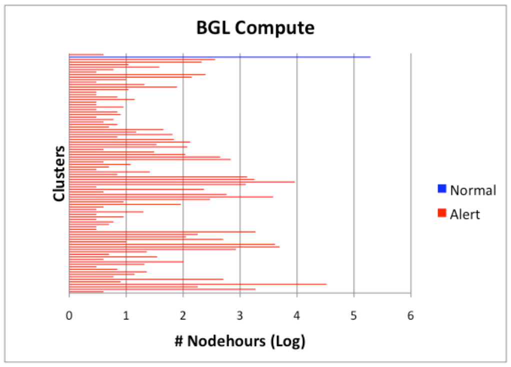
In the following sections the results on the *goodness* of the clusters formed using (i) internal measures (cohesion and separation) and (ii) ground truth (alert categories) are discussed. Finally, a discussion on how the results validate the hypotheses is given.

5.3.1 Internal Measures

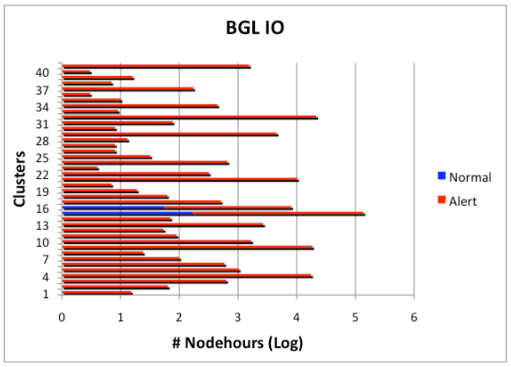
The results demonstrate that the clusters formed show an average Fwithin measure of ~0.01, indicating tightly formed clusters, and an average *Distance* measure of ~0.82, indicating that the clusters are well separated. Details of the Fwithin and *Distance* results for each of the datasets can be found in Table 5.3.

5.3.2 Ground Truth

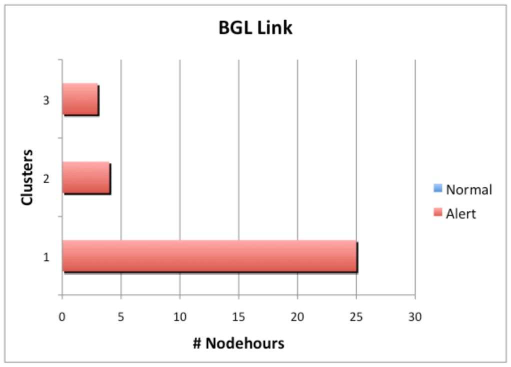
If the clusters formed have some sort of conceptual meaning then they should be able to separate alert behavior from normal behavior. This means that if at least one nodehour in a cluster is indicative of alert behavior then most of the other nodehours in the cluster should also be indicative of alert behavior. Results that highlight this for the datasets are shown in Figs. 5.6, 5.7, 5.8 and 5.9. These graphs indicate that the clustering method was able to separate normal nodehours from alert nodehours to a good degree for most of the categories. The degree of separation was not very pronounced with the Liberty-Admin, Liberty-Other, Spirit-Admin and Spirit-Other node categories. It should be noted that there are other clusters not shown in the graphs in Figs. 5.6, 5.7, 5.8 and 5.9 which consist entirely of normal nodehours.



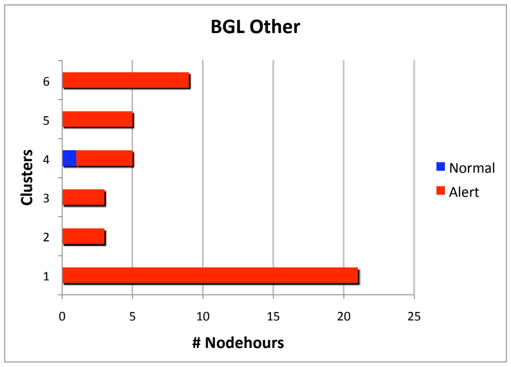
(a) BGL-Compute



(b) BGL-IO



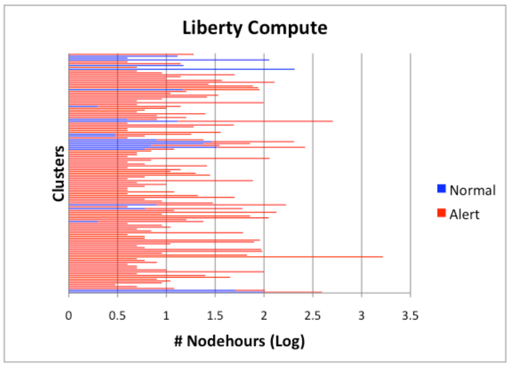
(c) BGL-Link



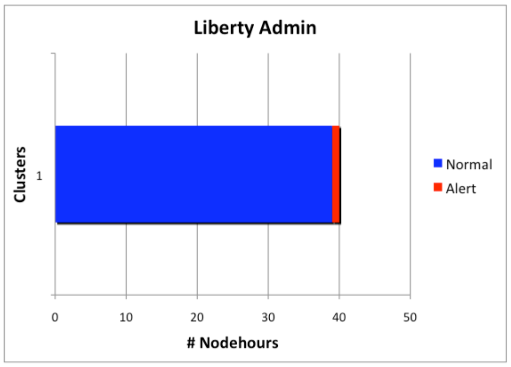
(d) BGL-Other

Figure 5.6: Stacked Bar Graphs for BGL node functionality categories. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). Only clusters which contain 3 or more nodehours are plotted.

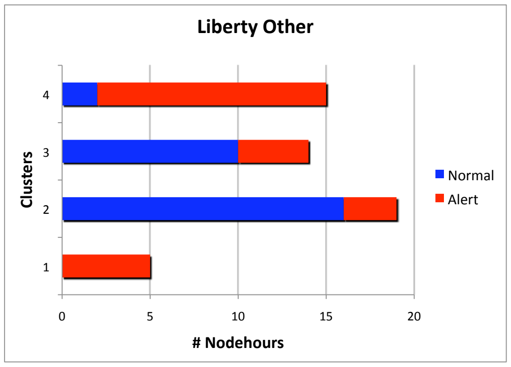
The provided ground truth gives not only the ability to differentiate alert messages from normal messages, it provides alert categories as well which allow the differentiation of different kinds of alerts. The conceptual purity ratio attempts to measure the degree to which the clusters are able to differentiate these alert categories, i.e. if one of the nodehours in a cluster contains the signature for a specific alert category, then to what degree do the other nodehours show the signature for that alert category. The results show an average conceptual purity ratio of ~0.96 with regard to the alert categories. Details are provided in Table 5.4. The first two columns of this



(a) Liberty-Compute



(b) Liberty-Admin

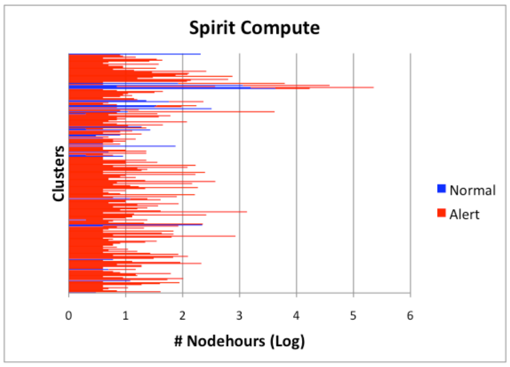


(c) Liberty-Other

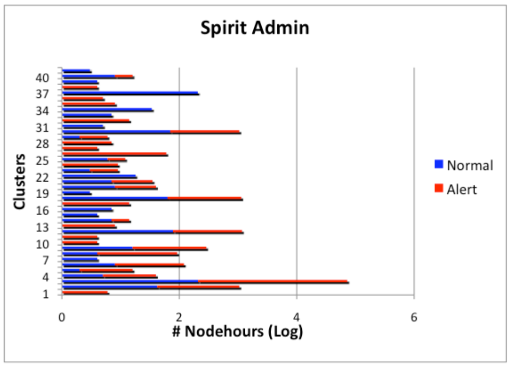
Figure 5.7: Stacked Bar Graphs for Liberty node functionality categories. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). Only clusters which contain 3 or more nodehours are plotted.

table present the number of alert categories (this is for the entire dataset and not just those found in the alert clusters) and the number of alert clusters, respectively.

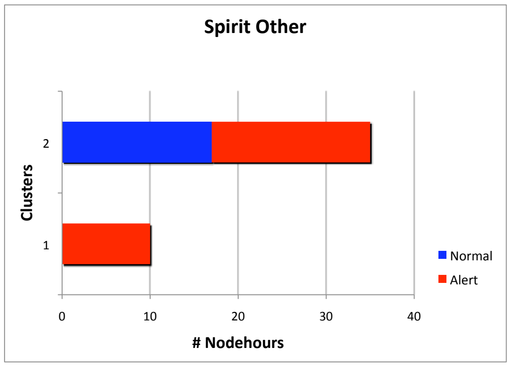
Since only clusters of size > 9 were considered, some of the datasets did not have any alert clusters of size > 9, hence no conceptual purity results are provided for these datasets. In addition, the results show that the relationship between alert categories and alert clusters did not follow a one-to-one correspondence in some instances. The signature of an alert category could be linked to several alert clusters, while an alert cluster could be linked with more than one alert category. Cases in which more than one alert category are linked to a cluster probably indicate the existence of correlated



(a) Spirit-Compute



(b) Spirit-Admin

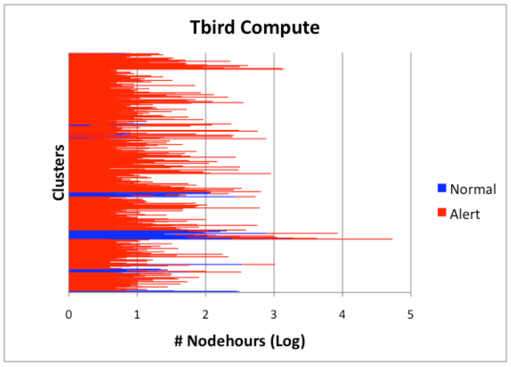


(c) Spirit-Other

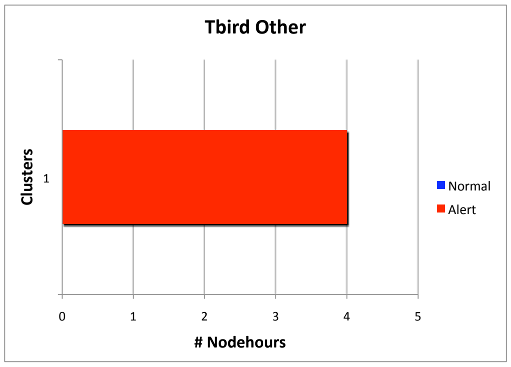
Figure 5.8: Stacked Bar Graphs for Spirit node functionality categories. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). Only clusters which contain 3 or more nodehours are plotted.

alert categories which occur together occasionally and an alert category being linked to different alert clusters could indicate different background activity at the point of occurrence. This background activity could indicate different causes for the same error. Overall, this highlights the complex interactions which go on in the system, with events showing different correlated behavior at different times. It is therefore an advantage that this approach can differentiate these different temporally dependent behaviors.

Due to the fact that only ground truth with respect to alert behavior was provided in the data, the evaluation measures only the degree to which the clusters formed



(a) Tbird-Compute



(b) Tbird-Other

Figure 5.9: Stacked Bar Graphs for Tbird node functionality categories. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represents the distribution of alert nodehours (red) to normal nodehours (blue). Only clusters which contain 3 or more nodehours are plotted.

mirror alert behavior. It is likely that results for alert behavior could be extended to normal behavior.

5.3.3 Discussion on Hypotheses

These results help to validate the hypotheses stated in Section 5.2.

- **Nodehours with the same ICS (based on Eq. 5.2), could contain the same unique set of message types :** The Fwithin results (i.e. Eq. 5.5) validate this hypothesis.

- **ICSs which occur frequently represent nodehours which contain strongly correlated message types and could represent some system behavior or characteristic :** The *Distance* (i.e. Eq. 5.5) and concept purity results validate this hypothesis.

5.4 Discussion of Results

This section starts with a discussion on the implications of the results and ideas on how this method can be generalized; it ends with directions which these results provide for the alert detection framework developed in the thesis.

The constraints on the distribution of message types across nodehours, which allow the method to work, are discussed in Section 5.2. These constraints have to be met for the proposed method to work. These datasets are from HPC systems which are different in configuration, installation location and usage. Despite these differences, the constraints applied to them all. This gives strong support to the idea that these constraints are generalizable to HPC systems and might even be to other systems which are similar, such as data-centres or cloud computing infrastructures.

The first constraint can be achieved by the utilization of sufficiently large datasets, i.e. a data set which has sufficient system log bins (spatio-temporal partitions). The second constraint is due likely to the Pareto sampling of messages and the choice of the time window for the spatio-temporal partitions used. Since the Pareto property is a characteristic of the log data, it is generalizable to most system logs. Therefore, meeting the second constraint would require only careful selection of an appropriate time window for the spatio-temporal partitions. In this research a one hour time window was utilized in the evaluations to generalize for all the datasets used. Previous work states that correlated messages in system logs could occur within time windows of between one second to one day [33]. A more generalized approach would be to vary this time window with the inter-arrival rate of messages in the system log. However, once patterns have been extracted from the system log, they can be applied without recourse to the time window which is used to extract them.

For the proposed method to work it is necessary for the events in a system log to be separable by source and time which is required for the information content calculations. This method cannot be generalizable for a system log in which source and time information is not available.

Is this method generalized to a system log which contains only messages from a single node or computer? The method is generalizable to this scenario because it is possible to leverage other system log fields to act as *sources* for the events in the log. Reporting processes or applications could be utilized if known; there is also the possibility of using time slices. In the use of time slices, the events are decomposed, so events produced during similar time slices (days of the week, off-peak, on-peak, weekdays, weekends, etc.) can be compared. The time slice should be chosen in a way which ensures that similar activity is performed. A proof of concept result for this is shown in Table 5.5. This table shows the results for running the proposed method on events produced by a randomly selected node from each of the *.Compute datasets. In this case, C in Eq. 4.2 is assumed to be the set of 24-hour periods in the data. The matrix $\mathbf{X}$ then represents a $|W| \times |C|$ matrix where $x_{w,c}$ is the count of the number of times term w was produced during a 24-hour period c.

It is noticed that appreciable separation of alert behavior from normal behavior was not achieved for the Liberty-Admin, Liberty-Other, Spirit-Admin and Spirit-Other node categories (see Figs. 5.7 and 5.8). This result might be due to the fact that while nodes in these categories are similar in terms of function, their logs are not similar. If this is true then a different means of defining similarity must be defined for these node categories. The work done with STAD does this, as STAD generalizes the approach to allow the definition of similarity even for cases in which nodes may not be similar. In addition, STAD takes advantage of the potential of this ICC method to separate alert states from normal states.

In determining patterns the proposed approach ignores the order in which messages occur in a nodehour. It might be interesting to investigate how taking this into consideration could affect the results. However, since clocks among several computers are not usually well synchronized to the level of precision which is required for such analysis, the correct ordering of the events is not guaranteed [77, 83]. Therefore checking to see if the ordering matters maybe futile.

The work presented in this chapter demonstrates how the decomposition of system log messages using source and time information naturally leads to the grouping of correlated message types within the partitions. Further, it demonstrate how these correlated message types can be discovered using the entropy-based information content of the partitions (nodehours) as a means of clustering the partitions. The method was evaluated using fourteen datasets derived from the functional decomposition of the system logs for the four HPC systems. The results show that the resulting clusters are well formed, i.e. having high internal cohesion and high external separation. The results demonstrate as well that, with a high level of confidence, it is possible to map *conceptually* the clusters to different alert categories. While the results are tested only on alert behavior, the proposed approach has the potential to be extended to normal behavior.

The well formed nature of the clusters produced by the proposed method lends credence to the assertion that these clusters mirror the different states of the system. The metrics used to evaluate the results, using ground truth on alert states and internal measures, provide sufficient evidence that these clusters are useable for describing the state of a system at a point in time. These states can be used to learn a Finite State Automaton (FSA) which mirrors the normal workflow of the system [15]. Hence the results have practical applications for enhancing the self-awareness and self-monitoring capabilities of a system which are important characteristics for an autonomic system [13].

Table 5.2: Cluster Statistics

	# Clusters (> 9)	**% Nodehours**	Total Clusters
BGL-Compute	135	99.96	446
BGL-IO	62	99.91	113
BGL-Link	6	97.71	17
BGL-Other	37	97.64	177
Liberty-Compute	412	99.74	3061
Liberty-Admin	202	87.82	2,069
Liberty-Other	345	88.86	5,326
Spirit-Compute	615	99.82	9,247
Spirit-Admin	259	86.53	2,118
Spirit-Other	495	78.46	8,641
Tbird-Compute	5,389	99.01	110,201
Tbird-Admin	557	94.01	3,492
Tbird-SM	78	68.62	2,000
Tbird-Other	556	97.65	8,653

Table 5.3: Average FWithin and *Distance* for Nodehour Clusters

	Avg. FWithin	**Avg. Distance**
BGL-Compute	0.000	0.963
BGL-IO	0.000	0.965
BGL-Link	0.000	0.914
BGL-Other	0.027	0.927
Liberty-Compute	0.000	0.748
Liberty-Admin	0.034	0.741
Liberty-Other	0.001	0.576
Spirit-Compute	0.003	0.784
Spirit-Admin	0.059	0.786
Spirit-Other	0.000	0.672
Tbird-Compute	0.000	0.896
Tbird-Admin	0.001	0.783
Tbird-SM	0.001	0.774
Tbird-Other	0.003	0.903

Table 5.4: Alert Nodehour Cluster Conceptual Purity

	# Categories	# Alert Clusters	Conceptual Purity Ratio
BGL-Compute	15	95	1.00
BGL-IO	15	41	1.00
BGL-Link	5	3	0.90
BGL-Other	6	6	0.90
Liberty-Compute	19	57	1.00
Liberty-Admin	3	0	N/A
Liberty-Other	9	1	1.0
Spirit-Compute	29	114	1.00
Spirit-Admin	3	9	0.86
Spirit-Other	6	2	1.00
Tbird-Compute	31	273	1.00
Tbird-Admin	7	0	N/A
Tbird-SM	0	0	N/A
Tbird-Other	10	0	N/A

Table 5.5: Cluster Measurements for a Random Single Node

	Avg. Fwithin	Avg. Distance	Conceptual Purity
BGL-Compute	0.00	1.00	1.00
Liberty-Compute	0.00	0.741	1.00
Spirit-Compute	0.00	0.734	N/A
Tbird-Compute	0.00	0.766	N/A

Chapter 6

Spatio-Temporal Alert Detection in HPC Logs

Following from the success of the evaluations of the ICC technique discussed in Chapter 5, this chapter discusses the details of the log alert detection method proposed by this thesis, i.e. Spatio-Temporal Alert Detection (STAD).

STAD is not a specific technique but a framework which allows flexibility in the choice of techniques used. There are three steps in the STAD framework. Firstly, it decomposes the events in a log spatio-temporally. "Spatial" refers to the source of the log event, e.g. a node in the HPC, and "Temporal" refers to the time the log event was reported, i.e. a timestamp. Secondly, it clusters the units resulting from the decomposition. If the clustering technique used in the second step is able to produce clusters which group different kinds of normal and anomalous activity together, then a third step is activated to separate the clusters which contain anomalous activity from those which contain normal activity, thus detecting the *alerts* in the anomalous clusters.

The specific techniques used in the implementation of the framework are discussed in this chapter. Nodehours are utilized as spatio-temporal units in this implementation. Other spatio-temporal units can be chosen - the choice is purpose and system dependent. Entropy-based ICC is utilized for the clustering of the nodehours, while a rule-based approach is utilized for anomalous cluster identification.

The main contributions of STAD are listed below.

1. The basic assumption for entropy-based alert detection is that *"Similar computers correctly executing similar code will have similar logs"* [55]. This work goes beyond this by assuming that *"System log events which are produced by similar spatial sources or produced during periods of similar system activity are likely to be similar"* [51]. The new assumption allows for the extension of the entropy-based approach to groups of dissimilar nodes.

2. STAD allows the determination of alerts without resorting to a ranking of spatio-temporal partitions.

3. A novel feature set is proposed for the identification of the anomalous clusters. The rules employed for identification use these features.

The evaluations of STAD on the HPC logs show that it is able to detect 100% of all alerts with a FPR of 0.8% in the best case, while achieving 78% Recall and a FPR of 5.4% on average.

STAD was compared to *NIPlus* which is based on nodeinfo [55]. The results of the comparison show that the average false positive rate achieved by *NIPlus* across the datasets is approximately 25% compared to 5.4% for STAD. An analysis of variance (ANOVA) test (at 5% significance) carried between the false positive rates achieved by *NIPlus* and those achieved by STAD show a statistically significant difference in favor of STAD.

The rules used in the third step of the implementation were developed manually. Since it is desirable for these rules to be generated automatically, a comparison between the results of using manually generated rules and automatically generated rules is provided. The automatically generated rules are generated by the C5.0 data-mining algorithm [62].

The material presented in this chapter can be found in these publications [50, 51].

6.1 Related Work

Alert detection is the main thrust of this thesis and related work has been discussed in Chapter 2. As STAD is the main contribution of this thesis in relation to alert detection, related work will not be discussed further.

6.2 Methodology

This section details the methods and techniques used in the implementation of the STAD framework. The STAD framework has three main steps listed below.

1. Spatio-Temporal Decomposition of log events

2. Clustering of Spatio-Temporal partitions

3. Identification of Anomalous Clusters

Each step of the process is general enough to allow for flexibility in the choice of methodology. The steps are described in more detail in the following sections.

6.2.1 Spatio-Temporal Decomposition of log events

The proposed alert detection mechanism is intended for use on very large and complex systems for which the manual inspection of system logs has become unrealistic. The system logs on such systems would contain information from the several components which make up the system. This is one of the properties which make system logs good indicators of system state. However, a single reported event in the system log is unlikely to be a good indicator of system state. Strongly correlated events in the log are more interesting and are better indicators of system state [63]. This problem was addressed in Chapter 5.

Previous work in log analysis have based their analysis on the correlated events in the log rather than on single events. Previous approaches to find correlated events in logs include frequent itemset mining [77, 36], tracking of variables reported in message types [83], time series analysis [57] and the PARIS algorithm [2]. One of the major obstacles to finding correlated events in system logs is the fact that correlated events may not always follow each other in sequence in the system logs [77]. To this end, the approach utilized here is the decomposition of the event log spatio-temporally.

Typically, events in system logs are not homogenous entities. Apart from the textual descriptions of the event, they contain information about the reporting component (source), which could be a hardware and/or software component, and the time of occurrence of the event (timestamp). By using the source and timestamp information to decompose the events in an event log such that each resultant unit of the event log contains only events from a single source over a unit of time, the chance that the events reported in such units are correlated is increased. Any combination of source and time information can be used for decomposing the contents of an event log spatio-temporally. However, only nodehours are utilized here.

6.2.2 Clustering of Spatio-Temporal partitions

Clustering can be described as the process of gathering entities into groups. The grouping is carried out in a way which ensures entities which fall into the same group are very similar but at the same time are very dissimilar from entities in the other groups [17]. The aim of this step of the process is to place the spatio-temporal partitions of the event log into partitions based on their similarity while minimizing the similarity among the eventual clusters. Hence, this step is referred to as a clustering phase. The information content-based technique [50] described in Chapter 5 is used in this step. Clustering using a traditional distance-based clustering technique is not feasible. The features used in clustering will be the unique terms (words) which appear in the log. The number of unique words in a log can number in the millions, leading to the curse of dimensionality. The ICSs used in this technique are derived from the terms which appear in the log after several steps of abstraction [50]. This work has shown previously that these scores provide an accurate means of finding the clusters without much computation.

Before the ICC of nodehours can be carried out, the entropy-based analysis of the contents of a log needs to be completed. This process has already been described in Chapters 4 and 5. However, for STAD, an extension of the process which allows its application to systems which contain dissimilar nodes is added. In Chapter 5 it was found that the ICC approach did not separate normal nodehours from anomalous nodehours satisfactorily in some node categories. It was conjectured that this occurred because the nodes in this category, while similar in function (similarity is defined by the function of the node), did not have similar logs. Consequently, a different means of clustering has to be found in such situations.

One way to approach the problem is to analyze the nodes in such dissimilar groups individually rather than as a group. In Chapter 5, proof of concept results were provided for how this can be done. In a case in which a group of functional nodes is considered dissimilar, the logs could be decomposed initially into separate logs so that each log is produced by only a single node. If single node log files are used, similarity can no longer be defined based on the source node and this thesis proposes that similar time periods can be used in this case. For instance, if the time period chosen is *days* then C, which represents source nodes in Eqs. 4.3 and 4.2 in Chapter 4 would correspond to each 24 hour period in the log. Hence, the events are now attributed to a *temporal* source, after which the ICSs can be assigned to the nodehours as usual. Please note that other time periods can be used here.

For clarification, the preferred method for assigning ICSs for this framework is *NiUniq* (see Eq. 5.2). This choice is made for the following reasons.

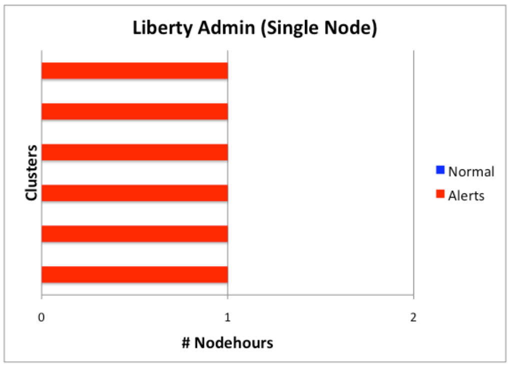
- **Fast Computation.** Due to its use of message types rather than raw terms in the log, *NIUniq* allows fast computation.

- **Higher Accuracy.** Evaluations have demonstrated that *NIUniq* seems to identify anomalous clusters more accurately than other techniques (see [48] and Appendix B).

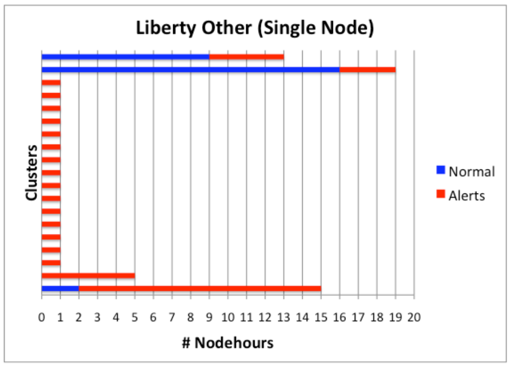
- **Clustering Value.** Due to the fact *NIUniq* considers all message types which exist in a spatio-temporal partition of a log, while ignoring their frequency of occurrence, it has more value for ICC than other techniques.

- **Temporal Filtering.** This proceeds from the previous item. Previous work has highlighted the importance of temporal filtering to log analysis [34]. By ignoring the frequency of the occurrence of message types, *NIUniq* performs implicit temporal filtering of system logs.

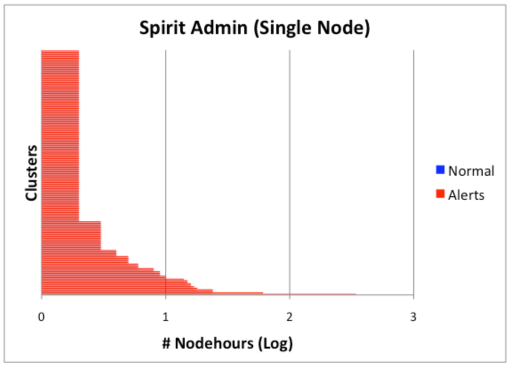
The proposed method for clustering dissimilar nodes was evaluated against the logs of the node categories which are considered dissimilar, i.e. Liberty-Admin, Liberty-Other, Spirit-Admin, Spirit-Other and Tbird-Admin. The results are promising. The charts in Fig. 6.1 show the stacked bar graphs of the anomalous clusters formed on a single node on each of these node categories. The clusters were derived using ICC, except the entropy-based analysis was carried out as described above. These charts indicate that the changes applied to the clustering method resulted in a better separation of normal nodehours from alert nodehours when compared to results obtained by assuming the nodes were similar.



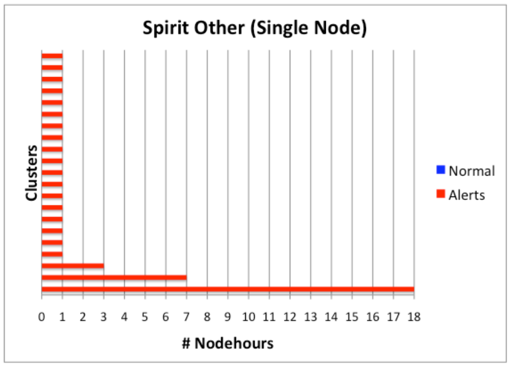
(a) Liberty-Admin



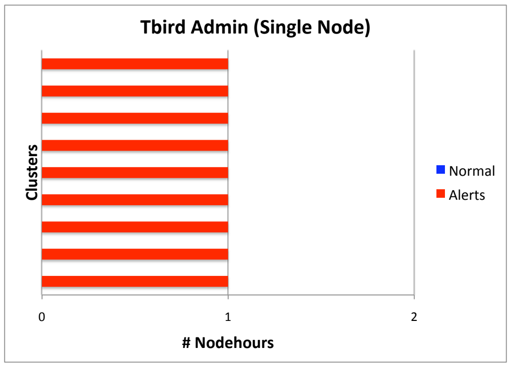
(b) Liberty-Other



(c) Spirit-Admin



(d) Spirit-Other



(e) Tbird-Admin

Figure 6.1: Stacked Bar Graphs for a single node from node functionality categories which are considered dissimilar. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). All clusters which contain at least one alert nodehour are shown.

While the stack graphs show how the clusters separate normal nodehours from alert nodehours, they do not show to which alert types the nodehours in the clusters should belong. If most of the alerts belong to one (or a few) alerts then the clusters can be said to define an alert state. This is illustrated using circos plots [31], which show the effectiveness of the technique in separating the different alert states in the system. A circos plot is a technique for visualizing the confusion matrix produced at the end of a classification or a clustering experiment. Typically, the rows/columns represent the actual-clusters/derived-clusters in a clustering experiment or actual-class/predicted-class in a classification experiment. For instance, the 2×2 confusion matrix in Table 6.1, which involves two actual classes (i.e. **A, B**) and two derived classes (i.e. **1,2**) will produce the circos plot shown in Fig. 6.2. The plot shows how each of the derived clusters map to the actual clusters using the values in the confusion matrix. An example of a more complex circos plot is shown in Fig. 6.3. This figure is the circos visualization of the alert clusters of the BGL-Link category. The figure shows the mapping between the nodehours in the four alert clusters (rows in the confusion matrix) derived using the ICC technique (i.e. $1, 2, 3, 4$) and the nodehours known to belong to the five alert categories identified by system administrators (columns in the confusion matrix) in the log data (i.e. $LINKBLL, LINKDISC, LINKIAP, LINKPAP, MONPOW$). This visualization shows that none of the derived clusters contains any $NORMAL$ nodehours and that a $1-1$ or $1-M$ mapping exists between the derived clusters and these alert category groupings (i.e. $1 \rightarrow LINKDISC$, $2 \rightarrow LINKPAP \,\&\, LINKIAP$, $3 \rightarrow MONPOW$ and $4 \rightarrow LINKBLL$). It is noted that the $LINKPAP$ and $LINKIAP$ alert categories are similar. Further explanations on how to interpret the circos plot are given below.

Table 6.1: Sample Confusion Matrix

Label	**A**	**B**
1	5	3
2	3	0

- **A:** These are arc shaped headers for row and column segments. There is an arc shaped header for each row and column in the confusion matrix. In the

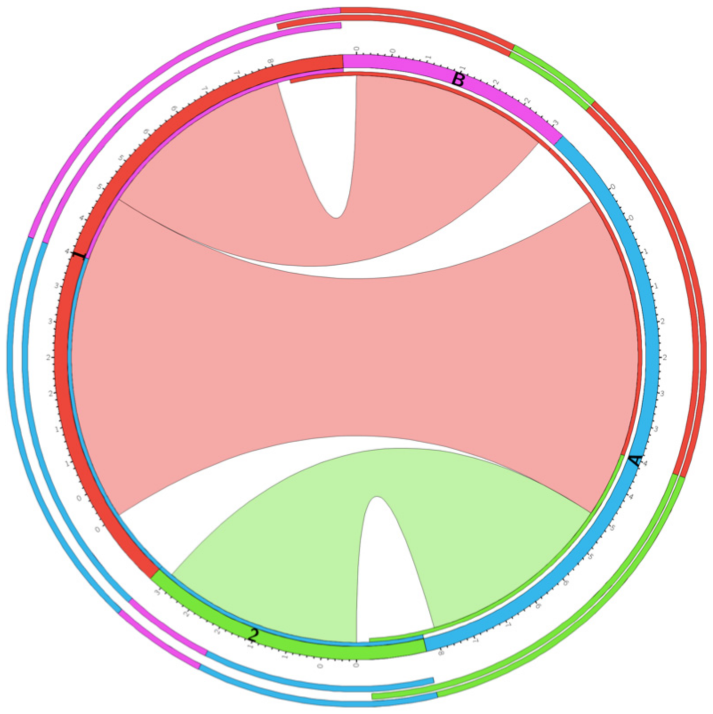


Figure 6.2: Simple 2 × 2 Circos plot example.

circos plots shown in this thesis, the row segments (derived clusters from the proposed system) are labelled using numbers, while the column segments (alert categories from the data sets) are labelled using the name of the alert.

- **B:** These ribbons show the mapping between row/column segment pairs. The size of the ribbon is proportional to the value for the row/column segment pair in the confusion matrix.

- **C:** Ribbons are colored either by the row or column segment they map from. In the circos plots in this thesis the ribbons are colored by row segments, i.e. the derived clusters they map from.

- **D:** These are ribbon ends. They are colored by the color of the row/column segment they map to. In this thesis, ribbon ends are colored by column segments, i.e. the alert categories they map to.

- **E:** This is a gap between a ribbon and its associated segment. The presence of the gap is another way of knowing whether the ribbon "maps to" or "maps from" the segment. The gap is present if the ribbon "maps to" to the segment and absent if it "maps from".

- **F:** These arcs encode the relative row, column and overall total values for the segments.

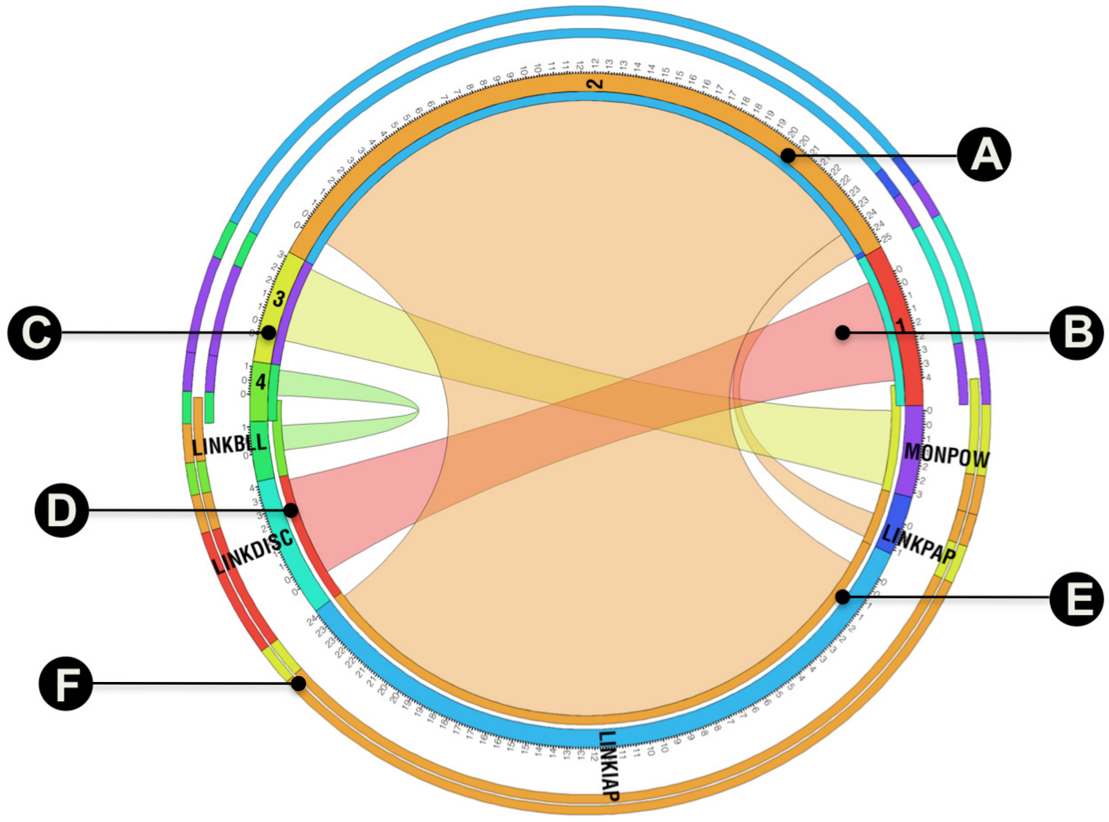


Figure 6.3: Example of a more complex Circos plot.

The circos plot visualizations of all the node categories are shown in Appendix C. Through inspection of these plots, alert detection is reduced to the task of identifying the derived clusters based on the possibility that they contain alert nodehours. This is what the third step of the proposed STAD framework attempts to achieve.

6.2.3 Identification of Anomalous Clusters

This step of the framework involves the separation of the clusters into two classes: an anomalous class and a normal class. Once a cluster is determined to be anomalous, all the nodehours which belong to that cluster are assumed to contain alerts. As with the other steps of the framework, any method of separation can be utilized.

To carry out the separation of the clusters, four important characteristics of alerts are identified. These are: Bursty, Endemic, Epidemic and Near-Periodic properties. Using these properties three assumptions about alert clusters which can be used in separating them from normal clusters are derived. An alert cluster refers to clusters which contain a majority of nodehours with alert signatures. These properties/assumptions and the identification rules implemented based on the assumptions are described in detail with real examples from the HPC logs in Section 6.3 and Section 6.4.

6.3 Cluster Separation

In this section, the four alert characteristics which were identified, are described (see Section 6.3.1). Based on these characteristics three assumptions about alert clusters are proposed (see Section 6.3.2). Finally, in Section 6.4, the rules derived from the assumptions are described as implemented in the evaluations.

6.3.1 Alert Characteristics

Four important alert characteristics were identified by going over time series plots of the activity for the different alert types identified in the log. They are described in detail below. It is noted that these alert properties are not mutually exclusive and are not intended to be exhaustive.

- **Bursty Property**. The bursty property occurs when there is a significant increase in the number of events reported over a period of time. An example of the bursty property is shown in Fig. 6.4. This figure shows the number of events (measured by size in bytes) produced by a single node (Ln30) on the Liberty cluster on an hourly basis over a 24 hour period. A significant increase in the number of bytes produced in the logs is seen in the 15th to 18th hours. Based on the labeling, there are at least 9 alert types active during this period, with the *R_EXT_CCISS*, *R_EXT_FS_IO* and *R_EXT_INODE*1 types being the major contributors to the burstiness experienced. The number of bytes produced by this node drops to zero in the 19th hour right up to and including the 24th hour. This indicates that the fault which caused this burstiness led to a failure of the node. This graph highlights the importance of alert detection in system management. The detection of this alert in the 15th hour would have given administrators a three hour window within which they could have applied remedial action to prevent the failure of the node which occurred in the 19th hour.

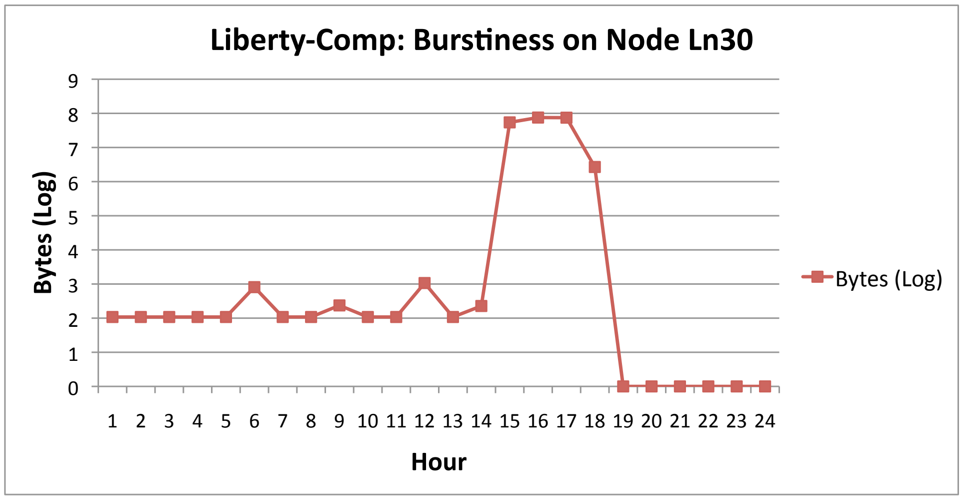


Figure 6.4: **Bursty Property:** This graph shows the size of the events produced by a single node from the Liberty HPC at hourly intervals over a 24 hour period.

- **Endemic Property**. The endemic property occurs when a cluster shows sporadic activity over a period of time as well as being localized at each occurrence. The graph in Fig. 6.5 shows the activity of a cluster from the BGL log. This cluster is associated with the *KERNMC* alert type. Occurrences of this cluster type are sporadic and affect only one node at each occurrence.

- **Epidemic Property**. The epidemic property occurs when a cluster shows sporadic activity over a period of time as in the endemic case but affects instead

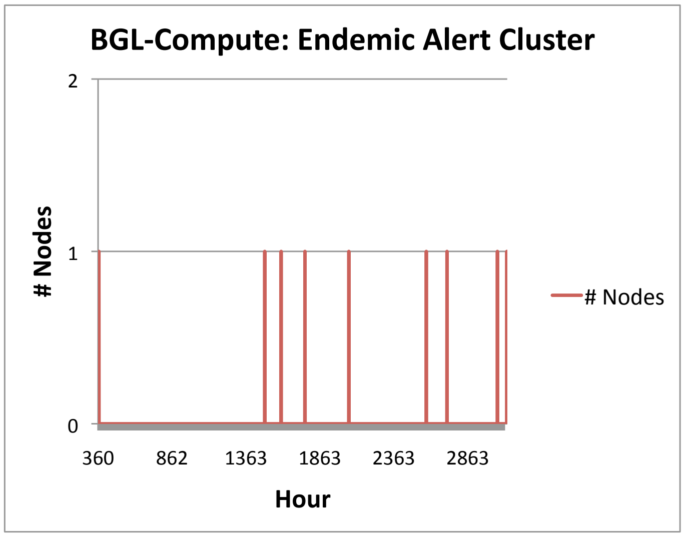


Figure 6.5: **Endemic Property:** The graph shows localized sporadic activity.

a relatively large number of nodes at each occurrence. The graph in Fig. 6.6 shows an example of another cluster from the BGL log which shows the epidemic property. This cluster is associated with the *KERNREC* alert type. Occurrences of this cluster type are sporadic and affect as many as $2,000$ nodes at each occurrence. Though wide spread, the activity does not affect all nodes: the total number of nodes in the BGL event log category to which this cluster belongs contains 65,554 nodes.

- **Near-Periodic Property**. The near-periodic property occurs when activity in a cluster occurs at almost regular intervals. The graph in Fig. 6.7 shows an example of a cluster exhibiting the near-periodic property. In this example from the Spirit event log the cluster type occurs almost on an hourly basis over a period spanning about four weeks. This almost regular rate of occurrence and the frequency of occurrence is an indication of the near-periodic property. The cluster shown in Fig. 6.7 is linked to the R_HDA_NR and R_HDA_STAT alert types.

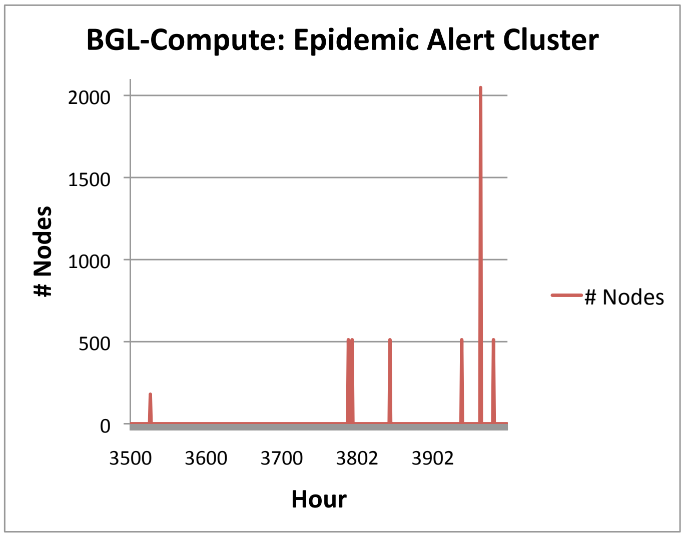


Figure 6.6: **Epidemic Property:** The graph shows sporadic activity which affects a relatively large number of nodes.

6.3.2 Alert Cluster Assumptions

This section describes the properties which the alert clusters are assumed to have. These assumptions are based on the identified alert characteristics and are the basis for cluster separation.

- Bursty, Endemic and Epidemic alert types show activity in relatively few time periods. This implies that the number of active periods for a cluster type should be a good indication of whether a cluster contains alerts or not. A cluster type which is active during relatively few time periods is likely to contain alerts. This supports the assumption that alerts are usually infrequent in an event log. The number of active periods is also a better measure of frequency than the count of active nodes or the count of nodehours.

- Endemic alerts show localized activity each time they occur. Measuring localized activity in a cluster would be a good indicator for an alert cluster. A cluster type which shows a high degree of localized activity each time it occurs is likely to contain alerts.

- Near-Periodic alert types have a frequent occurrence rate and may or may not be localized. Consequently, they may not be captured by the assumptions

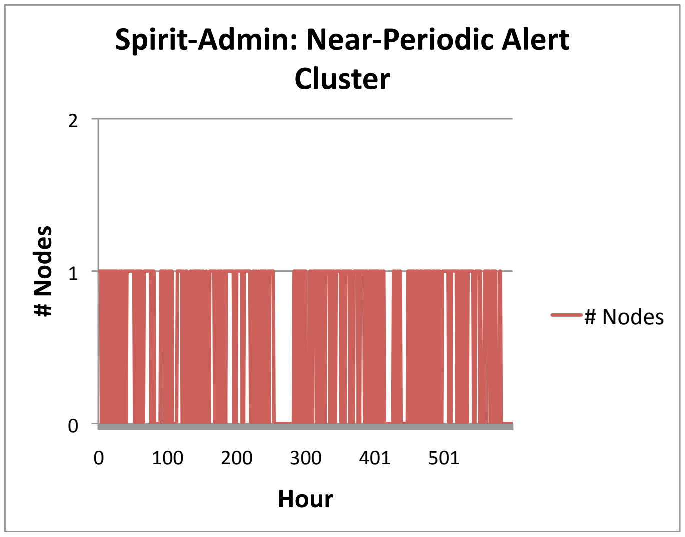


Figure 6.7: **Near-Periodic Property:** The graph shows very frequent activity compared to either the endemic or epidemic types and has an almost periodic rate of occurrence.

above. However, they do have the property of having close to regular rates of occurrence. Measuring the periodicity of a cluster would be a good indicator of this property. Consequently, clusters which are relatively frequent and show periodic or *almost* periodic activity are likely to contain alerts.

6.4 Identification Rules

The assumptions detailed in Section 6.3.2 were used as a means of identifying the nodehour clusters, which are derived from HPC logs. The details of the evaluation of the identification rules are described in Section 6.4.2. However, in this section, the implemented rules are described. There are three rules, one for each assumption. Before describing the rules, the following definitions are needed.

- Let E be the event log which we intend to analyze. Let each temporal period spanned in E be assigned an ordinal number, n. The first hour is assigned a value of 1 and every subsequent hour is assigned a value of $n + 1$ relative to its preceding hour, which would have a value of n.

- We define set $\mathbf{C}$ of spatio-temporal partition clusters derived from E, where $c_i \in \mathbf{C}$ is the ith element of $\mathbf{C}$.

- We define arrays $\mathbf{P}$ and $\mathbf{S}$, such that $\mathbf{P}[i]$ and $\mathbf{S}[i]$ are the counts of temporal periods and event sources reported in the nodehours in cluster c_i, respectively.

- For each cluster c_i, we define array $\mathbf{Q}_i$ such that $\mathbf{Q}_i[j]$ is the ordinal number of jth temporal period of activity for cluster c_i.

- For each cluster c_i, we define array $\mathbf{R}_i$ such that $\mathbf{R}_i[j]$ is the count of the number of event sources reporting activity of type c_i during the jth temporal period of activity for cluster c_i.

- $|\mathbf{Q}_i| = |\mathbf{R}_i| = \mathbf{P}[i] = m$

6.4.1 Rule Definitions

- **First Rule - Active Periods:** This rule implements the first assumption about alert clusters.

  1. Let med_per = Median($\mathbf{P}$), ignoring values where $\mathbf{P}[i] = 1$.
  2. For cluster c_i, if $\mathbf{P}[i] < med\_per$, then c_i is considered an alert.

  Due to the Pareto property, which is generally true for statistics involving system logs [77], several values in array $\mathbf{P}$ are $= 1$. Hence they are ignored in the calculation of med_per. If this is not done, then $med\_per = 1$, most of the time.

- **Second Rule - Localization:** This rule implements the second assumption about alert clusters.

  1. Calculate the average inverse node frequency INF_i for cluster c_i using Eq. 6.1.

$$INF_i = \frac{\sum_{j=1}^{m} \frac{1}{R_i[j]}}{m} \tag{6.1}$$

  2. Set an average inverse node frequency threshold INT.

3. For cluster c_i, if $INF_i >= INT$, then c_i is considered an alert.

   The average inverse node frequency metric, which is described in Eq. 6.1, attempts to provide a measure for localized activity. Its values are in the range $(0, 1]$ and the values close to one indicate localized activity within the cluster. The graph in Fig. 6.8 is a scatter plot of the alert clusters in the Liberty-Compute node category versus their average inverse node frequency values. The graph in Fig. 6.8 shows that most of the alert clusters have an average inverse node frequency value of 1 and therefore are showing the endemic property. In the implementation, INT is set to 0.95.

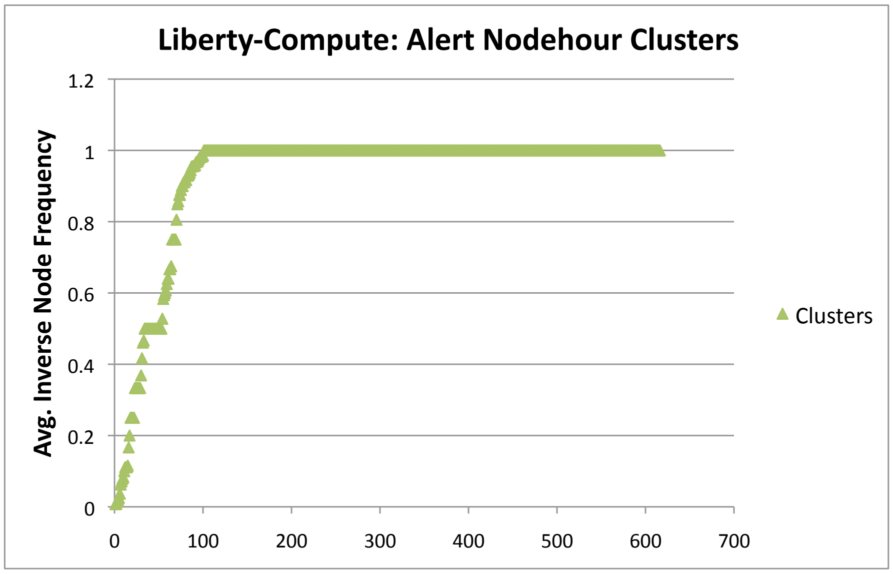


Figure 6.8: **Liberty-Compute Alert Clusters:** The graph shows a scatter plot of the alert clusters derived from the Liberty HPC log versus their inverse node frequency scores. The clusters are sorted based on their average inverse node frequency scores.

- **Third Rule - Periodicity:** This rule implements the third assumption about alert clusters.

1. Calculate the mean time between activity μ_i for cluster c_i using Eq. 6.2. μ_i represents the expected time between system activity of type c_i, if cluster

c_i was periodic.

$$\mu_i = \frac{\sum_{j=1}^{m-1}(Q_i[j+1] - Q_i[j])}{m-1} \tag{6.2}$$

2. Calculate the standard deviation STD_i from μ_i of intervals between system activity of type c_i using Eq. 6.3. It is assumed that STD_i values close to 0 indicate near-periodic activity.

$$STD_i = \sqrt{\frac{\sum_{j=1}^{m-1}[(Q_i[j+1] - Q_i[j]) - \mu_i]}{m-1}} \tag{6.3}$$

3. Set a standard deviation threshold STT.

4. For cluster c_i, if $STD_i < STT$, then c_i is considered an alert.

Let the *norm_per_cnt*$_i$ (normalized period count) for any cluster c_i be $\frac{\mathbf{P}[i]}{Max(\mathbf{P})}$. The bar graph in Fig. 6.9 shows STD for the Spirit-Admin clusters with a *norm_per_cnt* greater than 0.1. The Spirit-Admin node category is one of the functional node categories from the Spirit HPC log. Clusters with a *norm_per_cnt* greater than 0.1 represent those clusters with a relatively high number of active periods. Cluster 16 with a STD value of approximately 2 is the only cluster which contains alert nodehours. The adjacent clusters, i.e clusters 15 and 16, have STD values of approximately 1.8 and 4.0, respectively. This suggests that a STD close to zero is a good indicator for an alert cluster. In the implementation, STT is set to 2.5.

Since the near-period property requires frequent occurrences, this rule is only applied to mid-size clusters. *Size* is defined in respect to the count of active periods. Clusters with large period counts are left out, as they are likely normal. With this in mind, upper and lower bounds for the *norm_per_cnt* can be set. Then this rule can be applied only to clusters with a *norm_per_cnt* value which falls within these bounds. In the implementation the upper and lower bounds are set to 0.3 and 0.1, respectively. These values are set based on the Pareto property of event logs; clusters with mid-sized period counts would likely fall within these bounds.

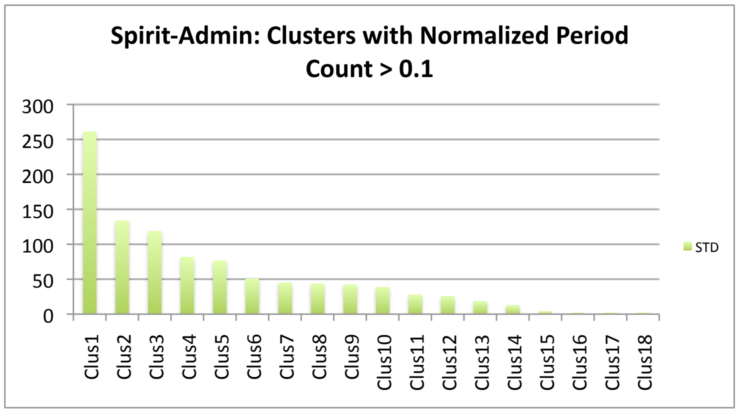


Figure 6.9: **Spirit Admin Clusters:** This graph shows a bar chart of clusters with a *norm_per_cnt* > 0.1 from the Admin functional node category of the Spirit HPC log. Cluster 16 is the only cluster which is known to contain alerts.

Algorithm 7 summarizes the procedure for identifying a cluster c_i as either an alert or a normal cluster using the rules above. It sets cluster c_i as an alert cluster if either of the rules above is true, and it sets c_i as a normal cluster if all of the rules are false. It is noted that the 2nd rule is not used when dealing with a dissimilar node scenario. The average inverse node frequency metric has no value for distinguishing alert clusters in such a scenario.

6.4.2 Evaluations

The evaluations include all the thirteen datasets listed in Table 4.7 in Chapter 4. The assumptions about the similarity of the node categories are highlighted in Table 6.2. A *Y* in the *Similar Nodes* column indicates that the dataset is processed under the assumption that the nodes in this dataset are sufficiently similar, while an *N* indicates that the nodes are assumed to be dissimilar.

In a real world scenario similarity of nodes would depend on function, configuration and usage patterns. Using these criteria it should be easy for a system administrator to make the decision of whether nodes are similar or not. In cases where it is difficult to decide, it is suggested that the nodes be considered dissimilar.

The values for all parameters were set as described in Section 6.4, except in the case of the *med_per* parameter. The implementation calls for the value of this parameter to be set automatically. It was found that the value assigned automatically to this

Algorithm 7 This pseudo-code describes the method for deciding whether a nodehour cluster contains alert nodehours.

Input: A nodehour cluster.

Output: Boolean value *is_alert*. 0 indicates a normal cluster while 1 indicates an alert cluster

```
1: $is\_alert = 1$
2: if $Rule1$ is $FALSE$ then
3:    if $Rule2$ is $FALSE$ then
4:       {$Rule2$ is only used when dealing with similar spatial sources.}
5:       if $Rule3$ is $FALSE$ then
6:          $is\_alert = 0$
7:       end if
8:    end if
9: end if
10: $Return(is\_alert)$
```

parameter is always in the range $[3, 5]$ on the datasets employed in this research. Based on this observation, experiments for each dataset where the value of *med_per* is set manually to values in the range $[2, 6]$ were run. This is in addition to experiments where its value was set automatically. The evaluations in which this parameter is manually set are compared against the auto-tune evaluations in Section 6.5.

The evaluation metrics used for the experiments are Recall (Detection) rate and FPR. These metrics are calculated using Eqs. 3.4 (in Chapter 3) and 6.4 (given below), respectively. The values for the TPs, FPs, TNs and FNs used in these equations are derived using the *binary scoring* metric as defined in [55]. In a dissimilar node scenario, processing is performed on a node-by-node basis, but during the evaluation, the TPs, FPs, TNs and FNs are summed to provide a single value for Recall and FPRs for the functional group.

To provide a baseline comparison, the proposed method is compared against *NIPlus*. As *NIPlus* uses a rank-based mechanism for alert detection, comparison is carried out only for FPR performance. The value of k (required for the Top_k separation of nodehours with *NIPlus*) which achieves a similar Recall rate as those achieved in the STAD experiments, is determined. The FPR is then calculated for *NIPlus* and

Table 6.2: Similarity Status of Node Functional Groupings used in Evaluations

	Similar Nodes
BGL-Compute	Y
BGL-IO	Y
BGL-Link	Y
BGL-Other	Y
Liberty-Compute	Y
Liberty-Admin	N
Liberty-Other	N
Spirit-Compute	Y
Spirit-Admin	N
Spirit-Other	N
Tbird-Compute	Y
Tbird-Admin	N
Tbird-SM	N
Tbird-Other	Y

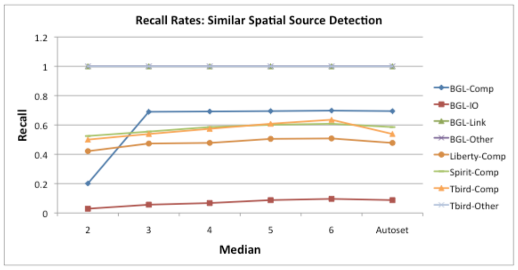
is compared to the FPR achieved using STAD.

$$FPR = \frac{FP}{FP + TN} \tag{6.4}$$

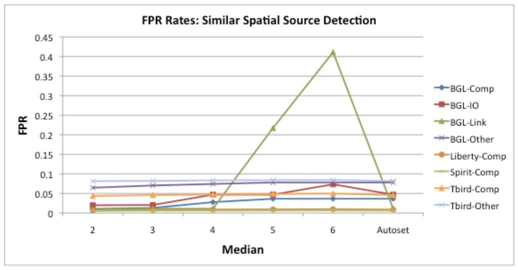
6.5 Results

For the similar node categories, 100% Recall is achieved in three node categories, i.e. *BGL-Link* , *BGL-Other* and *Tbird-Other*, irrespective of the value of *med_per*. Also, depending on the value of the *med_per* parameter, a Recall above 50% is achieved, with a single digit FPR for all node categories except the BGL-IO category (see Figs. 6.10a and 6.10b). With the *BGL-IO* node category a low 8% Recall was achieved. The reasons for this performance are explained in previous work [47]. In this node category, approximately 80% of the alert events are correlated closely to message type signatures which have entropy-based information content values, which are less than *0.1.* This indicates an almost equal rate of occurrence across nodes. This observation is due to the fact that certain error types in this category are not generated by the individual nodes but by an *IO* subsystem. Such errors are sensed and reported by all nodes. This means that these errors are attributed to the wrong *source* for the entropy-based analysis. The high alert nodehour ratio (38.22%) of this node category emphasizes the hypothesis that the alerts in this node category are unusual. If the results for this node category are adjusted by ignoring the alerts which show this property, the Recall increases to approximately 60%.

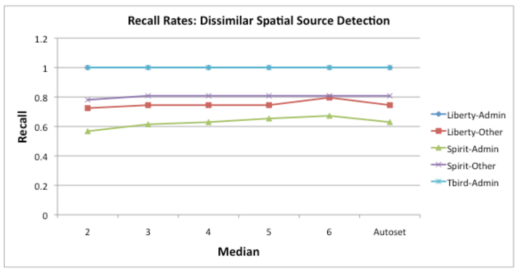
For the dissimilar node categories, 100% Recall was achieved in two node categories, i.e. *Liberty-Admin* and *Tbird-Admin*, irrespective of the value of *med_per*. Recall above 60% was achieved for all node categories, depending on the value setting of the *med_per* parameter. What is interesting here is that while it is noted that setting the *med_per* to 2 gives single digit FPRs for 4 out 5 node categories, the FPRs here tend to be on the order of about 1.5 times larger than those experienced with similar node alert detection. However, they are still better than what *NIPlus* achieves on the same data sets. This increase in FPRs may have to do with the simple method used for determining similar temporal sources, i.e. 24hr periods. A more sophisticated approach, which differentiates between actual days of the week, week-days, week-ends and so on, may yield better results.



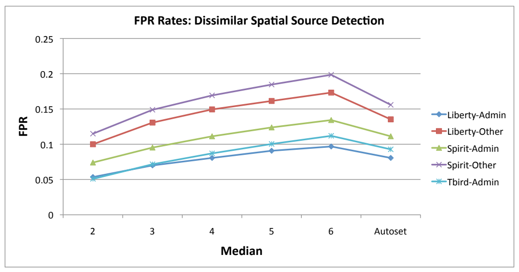
(a) Recall Rate - Similar Nodes



(b) False Positive Rate - Similar Nodes



(c) Recall Rate - Dissimilar Nodes



(d) False Positive Rate - Dissimilar Nodes

Figure 6.10: This figure shows recall and false positive rates for evaluations of STAD on the datasets listed in Table 4.7.

A summary of the results of the evaluation is given in Fig. 6.12. In this graph, the best case result for each dataset is selected from the experiments in which the value of the *med_per* parameter was set manually. These results are compared with the result in which the value of *med_per* was set automatically. The graph also shows a baseline FPR result using *NIPlus*. In choosing the best case a balance between achieving high Recall and a low FPR was considered. The graph shows that the overall best case was achieved with the *BGL-Link* category with 100% recall at a false positive rate of 0.8%. The average Recall (across node categories) was 78% and 77% for the manual experiments and auto-tuned experiments, respectively and the average FPR was 5.4%, 6.9% and 25% for the manual experiments, auto-tuned and *NIPlus* experiments, respectively. An ANOVA test carried out at 5% significance indicates that there is no statistically significant difference between the results achieved by setting the value of *med_per* manually and those achieved by setting it automatically. A similar test between the baseline false positive rates achieved by *NIPlus* against those achieved by the auto-tuned STAD results show a statistically significant difference. It is noted that had there been no statistically significant difference between the results, the observation that STAD achieves similar results without the manual determination of k for Top_k analysis, as employed by *NI* and NIPlus, remains a contribution of the method. In the determination of alerts through the information content ranking of spatio-temporal partitions by a Top_k analysis, it may be difficult to choose a value for k which can be used on all datasets. Detailed ANOVA results are provided in Appendix D. The ANOVA results are summarized in Table 8.1.

6.6 Anomaly Detection using C5.0

This section details the methodology and results for attempting to learn rules for identifying anomalous clusters automatically. The rules used in the previous evaluation were crafted manually. It would be interesting to see how automatically generated

Table 6.3: ANOVA Test Summary

Treatment	**F**	**P-Value**	**F crit**
FPR-Baseline vs. FPR-Autoset	5.036	0.034	4.259
FPR-BestCase vs. FPR-Autoset	0.003	0.957	4.259
DR-BestCase vs. DR-Autoset	0.817	0.375	4.259

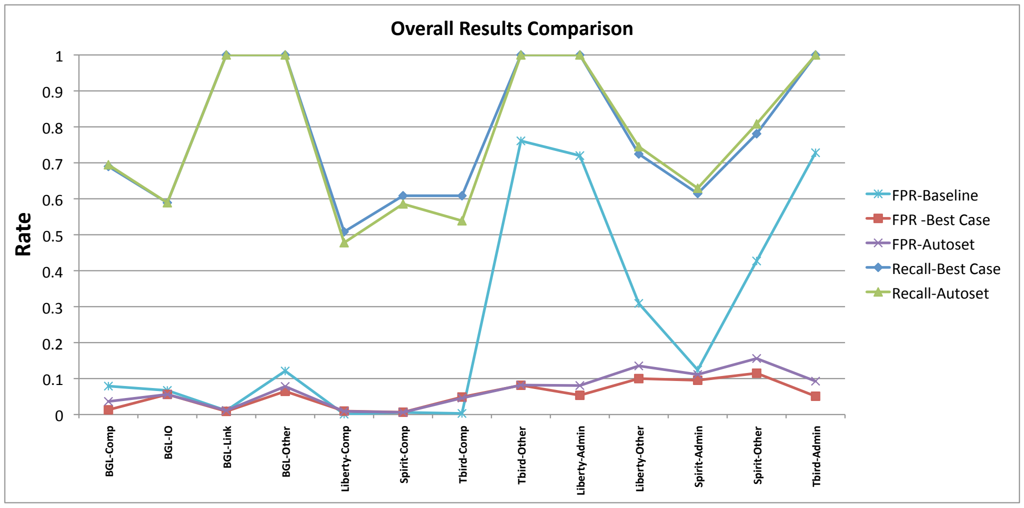


Figure 6.11: This graph shows the best case results for the experiments in which the *med_per* parameter is set manually and the results in which its value is set automatically. In addition, it shows the false positive rate achieved by *NIPlus* for a recall rate similar to the autoset experiments.

rules perform on this task. The C5.0 data mining algorithm is used in this evaluation [62].

C5.0 is the most recent version of the C4.5, which is derived from the ID3 tree induction algorithm [17, 62]. A decision tree is a tree structure which can be used in classification. Decision tree induction refers to the process of learning a decision tree from data. In a decision tree, each non-leaf node represents a test on an attribute, while each edge emanating from a node represents the possible values for said attribute or the possible outcomes of performing the test on the attribute. On the other hand, the leaf nodes take values from the set of possible classes to which the exemplars in the data can belong.

While the result of tree introduction is a tree structure, it is possible to represent the knowledge contained in the tree in the form of IF-THEN rules. Each path from the root node to all leaf nodes is a rule. The rule antecedent is the conjunction of the attribute tests carried out at each internal node along the path, while the rule consequent is the class label assigned to the leaf node in the path.

Classification of a data exemplar using a decision tree proceeds in the following manner: starting at the root, using the attribute tests at each node encountered, the attribute values of the sample are tested [17]. The sample finds a path through the tree by following the edge at each node which yields true when the test is performed on its corresponding attribute value. This is done until a leaf node is reached. The class value of the leaf node is assigned to the sample. The recursive algorithm for decision tree induction using the ID3 algorithm is outlined in Algorithm 8 [17].

Algorithm 8 This function, called *ID3_Tree_Induction*, generates a decision tree using a set of training exemplars. The function works recursively.

```
Input: A set of training exemplars, data and the set of attributes, attributes by which members of
    data are defined.
Output: A decision tree $t$ for classifying data
 1: Create a node $N$ of tree $t$
 2: Assign all elements of data to $N$
 3: if all elements of data belong to the same class, $C$ then
 4:     Label $N$ with class $C$
 5:     Return($N$) {$N$ becomes a leaf node}
 6: else if $|attributes| = 0$ then
 7:     Label $N$ with the most common class in data
 8:     Return($N$) {$N$ becomes a leaf node}
 9: else
10:     Select targetattribute from attributes using attribute selection criterion
        {Entropy-based Information_Gain is the most common criterion used. The attribute with
        the highest information gain is selected.}
11:     Label $N$ with targetattribute
12:     for every value, val that targetattribute can assume do
13:         Create a condition for the statement $targetattribute = val$
14:         Create an edge starting at $N$ for condition
15:         Let $data_{con}$ be the set of exemplars from data for which condition is $TRUE$
16:         if $|data_{con}| = 0$ then
17:             Create a node at end of the edge labelled with the most common class in data
18:         else
19:             $attributes_i = attributes \setminus targetattribute$
20:             $N_i = ID3\_Tree\_Induction(data_{con}, attributes_i)$ {$N_i$ is node}
21:             Attach $N_i$ to the edge
22:         end if
23:     end for
24: end if
```

C5.0 works in a manner similar to Algorithm 8. However, there are some enhancements to this basic algorithm and its previous successor C4.5 which are incorporated into C5.0 [59]. They are listed below.

- **Attribute Type.** ID3 tree induction expects all attributes it deals with to be categorical. If they are not categorical then they must be discretized before they can be used. C5.0 is generalized to deal with a whole range of attribute types, including continuous attributes.

- **Attribute Selection Criteria:** Since the $Information\_Gain$ metric is known to be biased toward attributes with a large number of values, C5.0 uses a range of more robust selection criteria for attribute selection (for example the *Gain_Ratio* statistic [17]).

- **Missing Values.** C5.0 has more robust ways of handling missing values. Classic tree induction handles missing values by replacing them with the most common value for the attribute.

- **Attribute Creation:** C5.0 has the ability to create new compound attributes from the attributes used to describe the exemplars in the data. This ability helps it to reduce problems such as fragmentation, (i.e. when the number of samples assigned to a node is too small compared to the original sample), repetition, (i.e. repeated testing of the same attribute along a branch of the tree) and replication, (i.e. the duplication of subtrees within the decision tree).

- **Attribute Winnowing.** This allows the algorithm to remove attributes from the attribute list if it considers them not to be useful.

- **Class and Attribute Weighting:** In classical tree induction all classes have the same importance. C5.0 allows classes to weight by their degree of importance. In this way, the tree induction pays more attention to errors resulting from the misclassification of exemplars which belong to classes with high weights.

- **Boosting.** The addition of boosting to C5.0 allows the induction of more accurate decision trees.

- **Complexity.** C5.0 uses less memory and is faster that either C4.5 or ID3. It also produces more compact decision trees.

For the evaluation the implementation of the C5.0 algorithm found at [62] was utilized. The full listing of parameters which can be used is detailed in Appendix A, while the parameter settings used for the evaluation are listed in Table 6.4. The "-b" option turns on boosting during tree induction. Each of the datasets was split into two in the ratio 3 : 1, with the larger split used for training the classifier and smaller split used for testing. Attention was paid to ensuring that the normal/anomalous cluster ratio remained the same in both the training and test files.

Table 6.4: C5.0 Parameter Settings

Parameter	**Value**
-r	On
-p	On
-e	On
-b	On

In the file each cluster was represented by the following features: #Nodehours, #Nodes, #Periods, ICS, Avg. Inverse Node Frequency, Mean Time between periods, Max Time between periods, Minimum Time between periods, Median Time between periods and Standard Deviation of time between periods from mean. This feature set includes all the features used in the manually generated rule set and more. This is appropriate as C5.0 has the ability to do its own feature selection.

The results of the evaluation were compared to the manually crafted rule-set in Fig. 6.12. The results show far more variability in the results obtained from C5.0 than that obtained using the manually crafted rule-set, indicating that the problem is a difficult one. C5.0 was unable to produce a usable classifier in at least five datasets but did produce classifiers which outperformed the manually crafted rules on two datasets for both Recall and FPR. However, it is noticed that the FPRs produced by the C5.0 rules tend to be much lower than those produced by the manual rules. An ANOVA test was performed to compare the C5.0 rules to the manual rules (see Appendix D). The results show a statistically significant difference in Recall rates in favor of the manual rules, while it shows a statistically significant difference in FPRs in favor of C5.0.

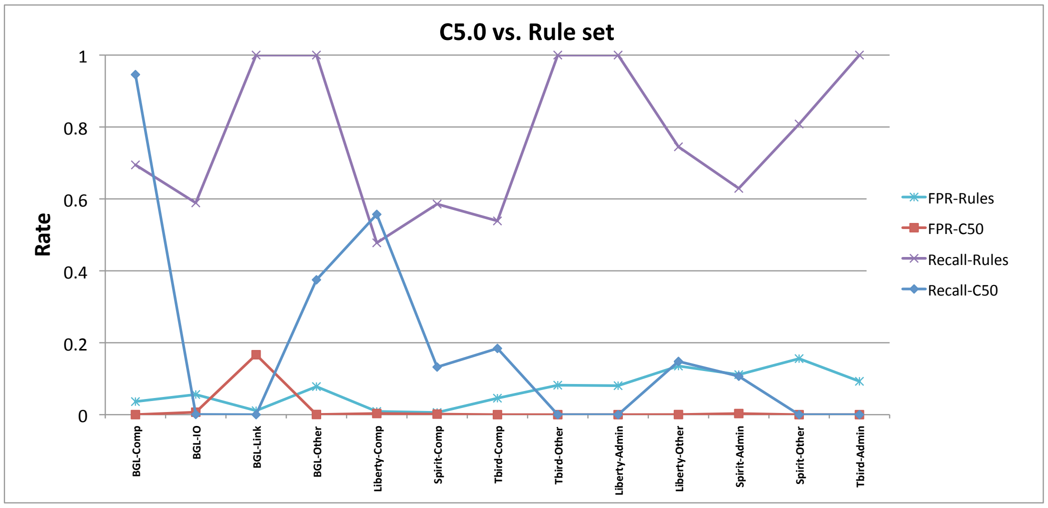


Figure 6.12: Comparing the Recall and FP rates for the default performance of C5.0 and the anomaly detection rule set.

Further tests were carried out to improve the results of C5.0 by varying its parameter settings. These tests were not successful. They are detailed in Appendix C.

6.6.1 Discussion of C5.0 Results

The results achieved by generating rules automatically using C5.0 allude to the difficulty of the anomaly cluster identification problem. However, while manually crafted rules seem to suffice, it is still important to be able to generate the rules automatically. Listed below are ideas about what may be done in the future to make automatic generation of the identification of rules successful.

- **Change the feature set:** Perhaps changing the feature set by adding more features may improve the results.

- **Use another algorithm:** Perhaps the use of another machine learning or data-mining algorithm other than C5.0 may lead to better results.

- **Reduce Class Imbalance:** The normal/anomalous cluster ratio in these datasets is skewed in favor of the normal clusters. Perhaps modifying the dataset to include more examples of anomalous clusters may improve the classifier's performance.

- **Model as a one-class problem:** The anomalous cluster identification problem is modeled as a two-class problem in the data presented to C5.O. Perhaps modeling the problem as a one-class problem may yield better results. It is noticed that the manually crafted rules form a one-class classifier. They identify only anomalous clusters and have no rules for identifying normal clusters.

- **Model as a multi-class problem:** The anomalous clusters belong to several different types of alerts. Perhaps training the classifier to identify the specific instance of an alert to which each cluster belongs may yield better results. However, this defeats the intended objective of training an anomaly detector.

6.7 Final Discussion of Results

The FPRs achieved by these experiments are not uncommon with anomaly detection systems [26]. The FPRs achieved are sufficiently low to support a semi-supervised approach. This would involve an administrator going over the detected alerts to document root causes and signatures for actual alerts and flagging signatures for the FPs. The system can use such information for future detection by searching for and reporting known alert signatures, thus suppressing future FPs. Such an approach will, lead to the reduction of the FPs to much lower levels. This approach has been used successfully with intrusion detection alarms [26].

Perhaps the most important lesson learnt from the experiments is that it is possible to separate clusters of event log spatio-temporal partitions which may contain anomalous activity from those which contain normal activity. The anomaly detection rules developed in this work demonstrate that separation of the clusters is possible. Even though minimal success was achieved with the initial attempt at automatic rule generation, it is safe to assume that if the rules can be generated manually then they can be generated automatically. This will be a viable project for future work.

The separation produced from such automatic identification might increase the accuracy further, as can be seen with the low FPRs achieved with the C5.0 classifiers. The instances of correlated message types, which appear in the clusters, are ordered temporally and thus form a time series. This suggests the potential that time series analysis techniques may prove useful during identication. This is worth exploring in future work as well.

In line with the goal of this thesis, the framework can be deployed easily with little or no user input. The only significant user input is the definition of the similarity categories used in the entropy-based calculations. The methods involved in the framework are computationally inexpensive and easy to understand. The overall accuracy of the framework could potentially be improved by working on improving the accuracy of each component of the framework. The majority of the techniques used in this implementation are unsupervised. Future studies could look at ways of incorporating user input into all of the phases of the framework, thereby improving accuracy through the use of semi-supervised techniques.

The extension of alert detection to dissimilar nodes provides the possibility of using this framework for alert detection on distributed systems. This is examined in the next chapter.

The application of STAD to production systems will involve the semi-supervised association of alerts to faults, i.e. converting anomalies to fault signatures. These fault signatures can be applied in an online setting for the detection of future occurrences of similar faults with lower false positives than those reported here. This would coincide as well with the proposal in [71] for the use of event messages in the goal of autonomic computing. This is the basis for the hybrid alert detection framework for production systems proposed in Chapter 1.

Chapter 7

Spatio-Temporal Alert Detection in Non-HPC Logs

The difficulty experienced in the manual analysis of system logs, due to size and complexity, is not limited to HPC logs alone. Cloud computing, i.e. the provision of computing as a service rather than as a product, is a paradigm being promoted in the computing world. This paradigm shift in the way computing power is provided calls for the installation of larger and more complex data centers. Even on the enterprise scale, the size and distributed nature of certain enterprise networks make not only the analysis of logs difficult but even the collection of such logs. This has led certain cloud providers to offer Logging-as-a-Service (LaaS).

The ability to extend an alert detection framework such as STAD to these types of logs would prove useful. This chapter provides details of evaluations which demonstrate not only that STAD can be used on logs from the kinds of infrastructure mentioned above, but also the flexibility of STAD as an alert detection mechanism. In this chapter, the evaluations are carried out using two new datasets, one from a distributed system and the other from a private cloud.

7.1 Methodology

Two system logs are used to ascertain the usefulness of the STAD framework for log files collected on non-HPC logs, i.e. a distributed log and a cloud log. The statistics of these log files are detailed in Table 7.1 which shows that these logs contain log information from multiple nodes and multiple applications (processes).

The distributed log comes from a cloud-based logging platform, i.e. a LaaS provider. The service provides a centralized system for the collection of enterprise log information and provides a web-based interface to view and analyze the information remotely. The log contains information from several independent machines located around the globe. As asserted by the provider, the log contains information from machines located in Africa, Asia, Europe and N. America. Most of these machines are configured differently, have different functions and are not aware of each other's existence. They are random machines which have been configured to submit their log data to a remote server located at the site of the LaaS provider. While the distributed logs come from a cloud-based service, they cannot be referred to as cloud logs as they do not come from the machines hosting the cloud services. On the other hand, the cloud log comes from a private cloud located within the network of an enterprise which provides web farming services. The logs come from the blade servers, five in number, on which the actual cloud services are hosted. Hence, they are referred to as cloud logs.

Table 7.1: Non-HPC Log Data Statistics

Log	**Days**	**Size(MB)**	**# Events**	**#Nodes**	**#Applications**
Distributed-Log	15	89.7	616,163	26	52
Cloud-Log	120	2109.52	11,294,552	5	46

The most important question which needs to be answered before STAD can be applied to these logs will be how to define *similarity*. For the HPC logs it was defined using the functionality of the nodes. In the case of the distributed logs, the nodes in the logs were all independent and did not serve the same function, hence functionality could not be used to define similarity. However, after exploring the logs, it was discovered that some of the machines run similar applications. Hence, for the distributed logs, similarity was defined on an application basis. Five applications, which were used on more than four machines and had a significant number of log entries were selected for analysis. They are listed below.

- **Avahi.** This is a service which allows devices to connect and use services on an IP network without administrator configuration.
- **Kernel.** The *kernel* is a service at the core of most computer operating systems (OS). It serves as an intermediary between each software application running on the machine and the machine's hardware.
- **Sendmail.** This is a service which enables the routing of email messages between different networks and it is not linked to any specific email delivery protocol.

- **SMBD.** This stands for Samba daemon and is one of the daemons associated with Samba. It provides services which allow file and printer sharing between computers running Windows-based OSs and computers running UNIX-based OSs.

- **SSHD.** This stands for Secure Shell (SSH) daemon. SSH is a network protocol which allows for secure communication between computers via an encrypted channel which may pass through an insecure network. SSH is used for data transfer or remote command execution.

The statistics of the resulting datasets are highlighted in Table 7.2. The nodehour was maintained as a unit for spatio-temporal decomposition. It is important to note that these nodehours are slightly different from the nodehours used for the HPC logs. More appropriately these nodehours should be referred to as app-nodehours (application nodehours), as they contain one hour of log information from a single node and a single application. By contrast, the nodehours used for the HPC logs (previous chapter) contained one hour of log information from a single node and several applications.

Table 7.2: Distributed-Log Dataset Statistics

	# Events	**# Nodes**	**# App-Nodehours**	**% Alerts**	**# Msg_Types**
Avahi	113	5	16	12.5	15
Kernel	2129	12	74	10.8	344
Sendmail	263	8	29	58.6	13
SMBD	2505	10	129	13.2	12
SSHD	45,776	12	212	15.1	16

In the case of the cloud logs, two of the servers had the same function of hosting the cloud virtual machines (VMs). These machines were placed in a cluster so that the VMs could use resources on both servers and could be migrated easily from one server to the next. The initial thought was to select these machines and analyze them as was done with the HPCs. In order to make the analysis different in this case, it was decided that app-nodehours be used instead of nodehours. Three applications, listed below, are selected.

- **Messages.** This application log contains general and system related messages from applications running in a Linux-based environment. This log is located in the /var/log/message sub-directory on most Linux-based OSs.
- **Secure.** This application log contains security related messages from applications running in a Linux-based environment. This log is located in the /var/log/secure sub-directory on most Linux-based OSs.
- **Xen.** Xen is a hypervisor application. It allows multiple virtual machines running different OSs to execute on the same hardware simultaneously.

The statistics for the resulting datasets are highlighted in Table 7.3.

Table 7.3: Cloud-Log Dataset Statistics

	# Events	**# Nodes**	**# App-Nodehours**	**% Alerts**	**# Msg_Types**
Messages	1,878,572	2	886	4.9	152
Secure	567,283	2	982	100	34
Xen	391,951	2	980	6.9	319

Upon performing entropy-based analysis on the messages found in the datasets listed in Table 7.3, it was found that ~95% of message types had entropy-based ICS of exactly 0. This indicates that ~95% of the message types had an exactly equal rate of occurrence across both nodes. This is not only odd but it would make the usefulness of the results of further analysis doubtful as well. Further investigation revealed that a large portion of the activity carried out in the chosen applications was controlled by the cloud hypervisor, hence they did not operate independently. Therefore, the approach of defining similarity was changed to the approach used for dissimilar nodes in the HPC case, i.e. each node was treated independently from the other. When this approach was taken, the percentage of message types which had entropy-based ICS of exactly 0 dropped to 0%.

As these logs did not have *alerts* defined and labelled by system administrators as was the case with the HPCs logs, alerts were defined naively by labeling as an alert a log event which contained any of the following terms: *ERROR*, *FAIL* and *WARN*. While it is acknowledged that this approach may end up including non-alerts as alerts and vice versa, I believe that it is sufficient for a proof of concept.

The entropy-based calculations and ICC were carried out using the exact same procedure as that used with the HPC logs. Entropy-based calculations for the distributed logs followed the same procedure used for the similar node HPC datasets, while the procedure for the dissimilar node datasets was used for the cloud logs. The rules and parameter settings used for anomaly detection remained the same. The results of the evaluations are presented in the next section.

7.2 Results

The result of the ICC of the logs will be highlighted first. Scatter plots of the ICSs of the nodehours derived from the datasets are shown in Figs. 7.1 and 7.2. From these graphs the phenomenon of strong clustering of spatio-temporal partitions around ICS values can be seen. This lends credence to the assertion that this phenomenon is not isolated and occurs in non-HPC logs as well. This result makes the use of the ICC technique a viable option for these datasets.

The results of evaluating the clusters formed after ICC are highlighted in Tables 7.4 and 7.5. The results show an average Fwithin of ~0.01, distance of ~0.748 and conceptual purity ratio of 1.00 for the distributed log datasets, while an average Fwtihin of 0, distance of ~0.440 and conceptual purity ratio of 1.00 for the cloud log datasets was recorded. These values indicate that the technique works appreciably with these datasets. The clusters are well formed and are sufficiently dissimilar from each other. Hence, they may represent distinct system states. It was found that by considering only clusters that had more than nine nodehours in them, it was possible, on average, to cluster ~80.78% and ~80.05% of all nodehours in the distributed and cloud log datasets, respectively.

The stacked bar graphs in Figs. 7.3 and 7.4 show the distribution of nodehours in the alert clusters formed after ICC. Just as is the case with the HPCs, it was observed that the resulting clusters separate normal nodehours from alert nodehours to a great degree. Hence, the identification of anomalous clusters for detecting alerts, as is done with STAD, is a possible next step.

After the identification of the clusters using the rule set, it was found that an average Recall of ~62.2% and average FPR of ~5.9% was achieved for the distributed log datasets, while an average Recall of ~57.5% and average FPR ~5.8% was achieved

Table 7.4: Average FWithin, Distance and Conceptual Purity for Nodehour Clusters from the Distributed Log

	Avg. FWithin	Avg. Distance	Conceptual Purity Ratio
Avahi	0.0	1.0	1.0
Kernel	0.0	0.83	1.0
Sendmail	0.03	0.61	1.0
SMBD	0.0	0.61	1.0
SSHD	0.02	0.69	1.0

Table 7.5: Average FWithin, Distance and Conceptual Purity for Nodehour Clusters from the Distributed Log

	Avg. FWithin	Avg. Distance	Conceptual Purity Ratio
Messages	0.0	0.62	1.0
Secure	0.0	0.46	1.0
Xen	0.03	0.24	1.0

for the cloud log datasets. Detailed results are highlighted in Figs. 7.5 and 7.6. While these results are respectable, it will be noticed that low Recall rates were achieved in three datasets: *Sendmail*, *SSHD* and *Secure*, where Recalls of 17%, 34% and 10% were achieved, respectively. Based on these results, it was thought that perhaps investigating with parameter values might yield better results for these datasets.

In the case of the *SSHD* log, it was found that increasing the size of the upper bound parameter from 0.3 to 0.4, effectively increasing the range of clusters which are checked for the near-periodic property, increases the Recall to 78% from 34% with no corresponding increase in the FPR. This approach worked for the *SSHD* log because it contained a large alert cluster which demonstrated the near-period property. The previous setting of the upper bound parameter excluded this cluster from being examined for the near-periodic property. This approach did not work for the *Sendmail* and *Secure* datasets. However, both these logs had at least one very large alert cluster which contained at least 50% of all alerts, see Figs. 7.1 and 7.2. This makes it difficult to achieve high Recall rates without the detection of these large clusters.

It is thought that changing the dimensionality of the decomposition could solve this problem. This approach would have the effect of separating alert behaviour further from normal behaviour, leading to smaller alert clusters. This assertion makes a lot of sense for the *Secure* dataset especially. In this dataset, all the clusters contain alert behaviour, meaning that at least one event containing terms relating to alert behaviour is reported in every hour of activity in the log. Using a smaller unit of time for decomposition might help separate this behaviour. Therefore the use of minutes for spatio-temporal decomposition was therefore investigated.

The graphs in Fig. 7.7 show the scatter plots for the ICSs for these datasets when app-nodehours and app-nodeminutes were used as a means for spatio-temporal decomposition. An app-nodehour is a spatio-temporal decomposition unit which contains one hour of log information from a single application running on a single node. An app-nodeminute contains one minute of log information from a single application running on a single node. The graph shows that the aforementioned assertion i.e. the reduction in the granularity of spatio-temporal decomposition by using minutes instead of hours, may lead to the production of smaller alert clusters is plausible. The newly pronounced separation of alert behaviour from normal behaviour in the *Secure* log dataset is of particular note. Performing cluster separation using clusters of nodeminutes increased the Recall marginally to 19% from 17% with no increase in FPR for the *Sendmail* dataset, while an increase of Recall to 14% from 10% with only a negligible increase in the FPR (0.06% from 0%) is recorded for the *Secure* dataset. Though the results improved, the marginal increase indicates that some large clusters remained undetected.

In the case of the *Secure* dataset, it was discovered that the large cluster exhibited the near-periodic property. However, the value of the STT parameter was unable to identify it as being anomalous. The STD values in this dataset seemed on average larger than usual. Hence, by changing the value of STT to 14, it was possible to increase the Recall to 73% without a corresponding increase in the FPR. However, no success was achieved in this respect for the *Sendmail* dataset. Given the naive method of identifying alerts in these datasets, it remains to be seen whether the large alert cluster in this dataset represents true alert behaviour. It is noted that this alert cluster was the largest cluster formed, which is odd for an alert cluster.

7.3 Discussion

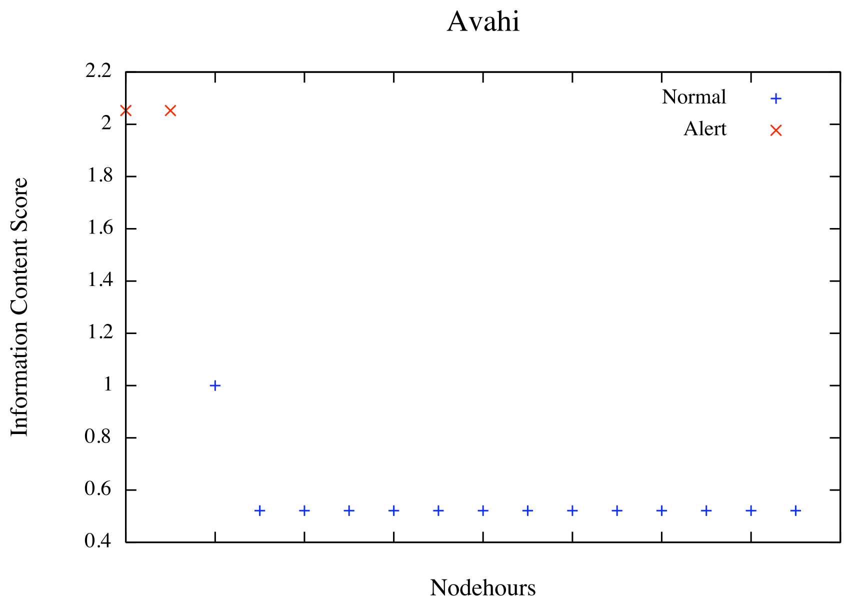
The evaluations presented in this chapter demonstrate in practical terms the versatility of the STAD framework, having experimented with different spatio-temporal partitioning approaches using nodes, applications, hours and minutes. In the task of alert detection in system logs one size rarely fits all. However, STAD is flexible enough to be adopted for the specific needs of the infrastructure on which it is to be deployed. This is definitely an advantage of the framework.

With the inclusion of the distributed logs and cloud logs, it was demonstrated that STAD is not limited to use on HPC logs alone. A large spectrum of ways (listed below) of defining similarity in log files has been covered.

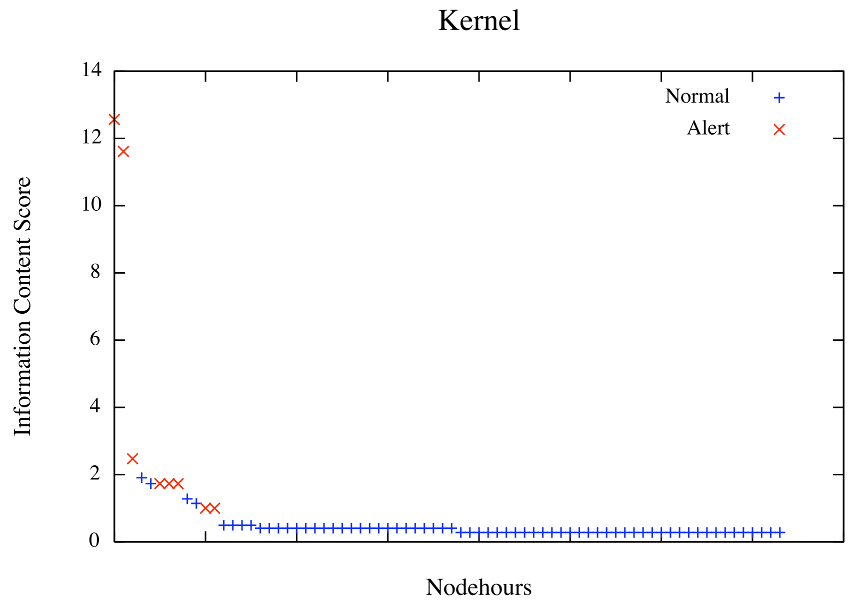
- Multiple applications running on multiple nodes was covered by the evaluations of the similar HPC logs.
- Multiple applications running on a single node was covered by the evaluations of the dissimilar HPC logs.
- Single application running across multiple nodes was covered by the evaluation of the distributed logs.
- Single application running on a single node was covered by the evaluations of the cloud logs.

The evaluations have revealed that while similarity is important for meaningful entropy-based analysis, a relative amount of independence between the similar entities is necessary for meaningful analysis to take place. Performing entropy-based analysis on entities which are not independent will yield models that indicate that all behaviour(s) is/are normal.

The importance of parameter settings was highlighted as well. The fact that STAD has these parameters is an advantage but users not understanding them could be a disadvantage. Investigations into methods for setting the values of these parameters automatically will make for interesting future research.

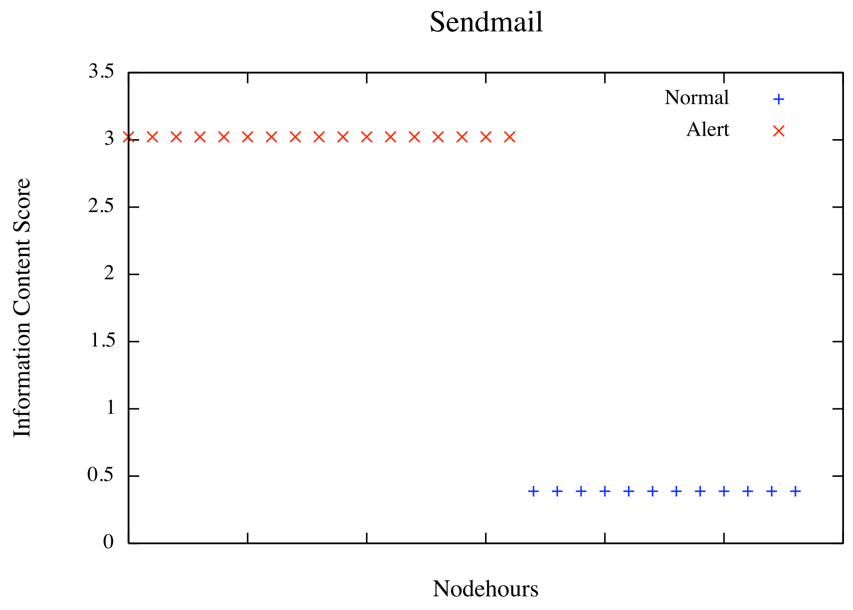


(a) Avahi

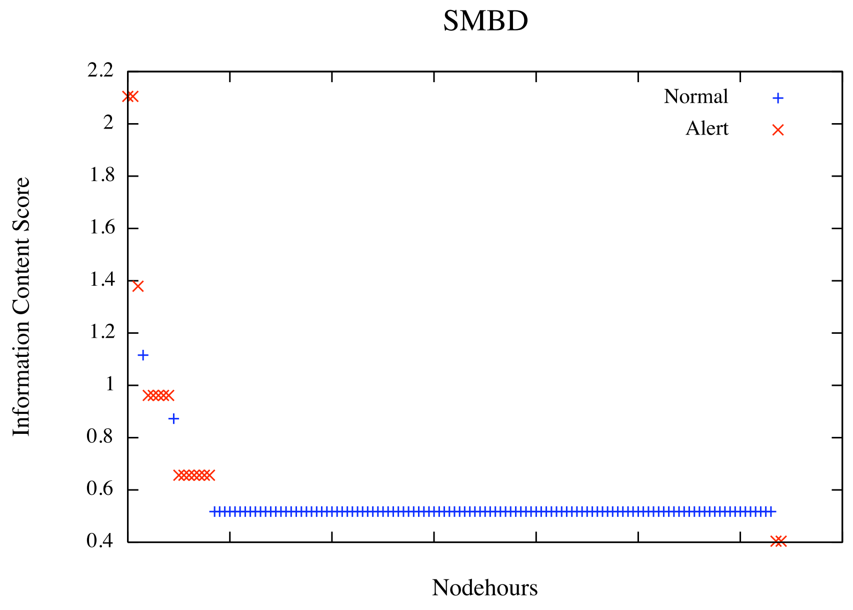


(b) Kernel

Figure 7.1: Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the distributed log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot.

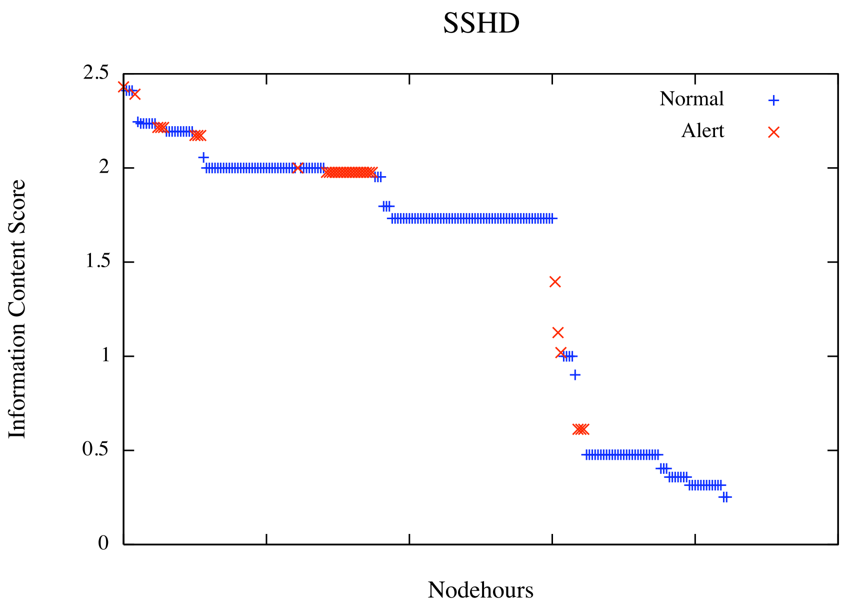


(c) Sendmail



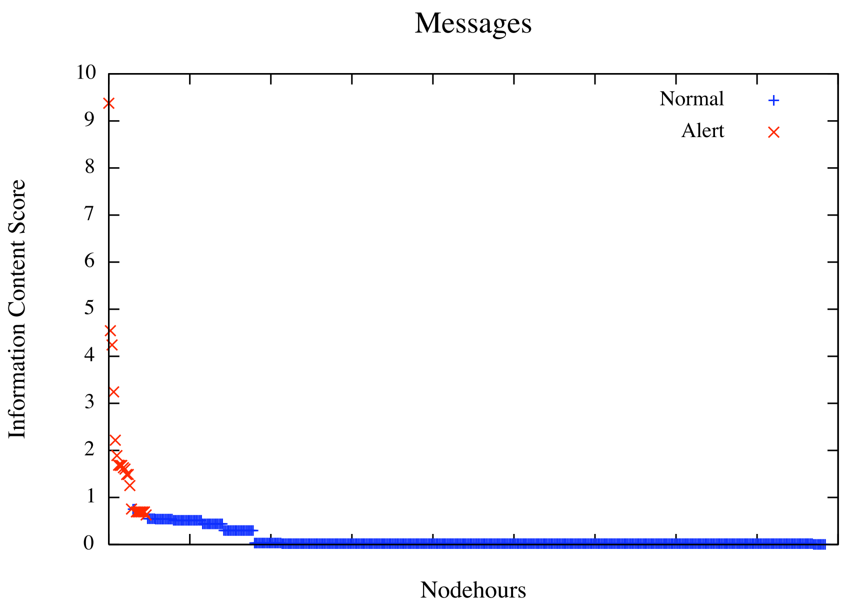
(d) SMBD

Figure 7.1: Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the distributed log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot.

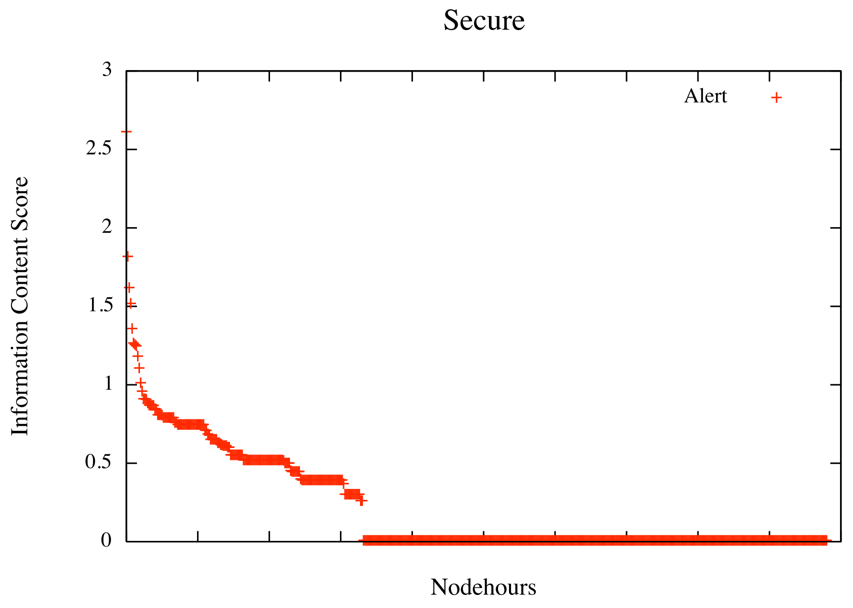


(e) SSHD

Figure 7.1: Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the distributed log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot.

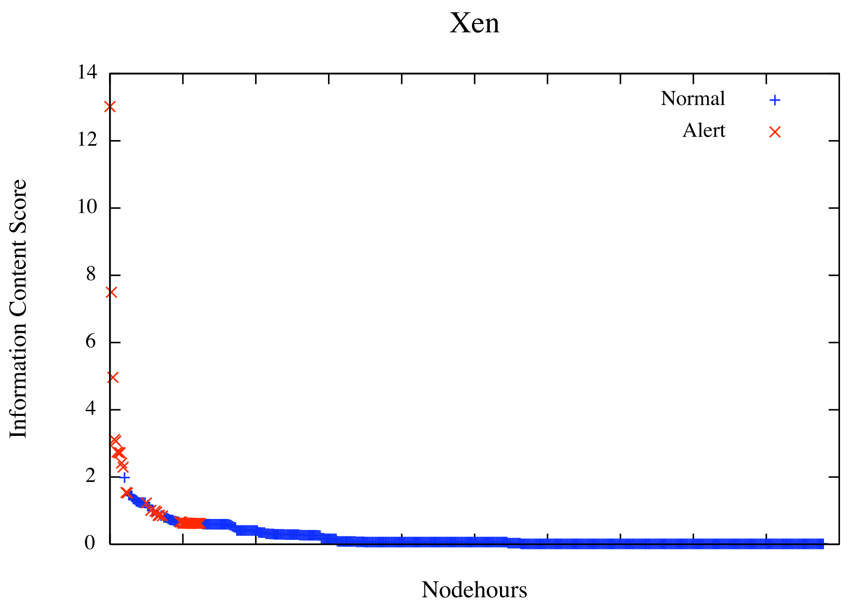


(a) Messages



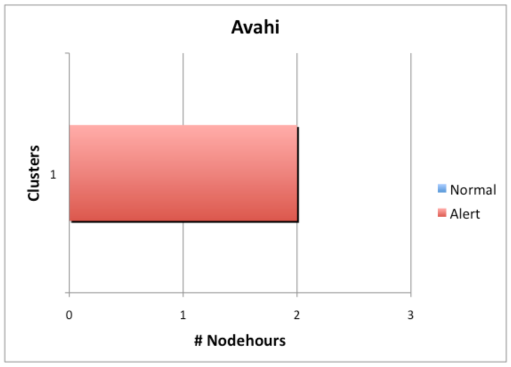
(b) Secure

Figure 7.2: Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the cloud log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot.

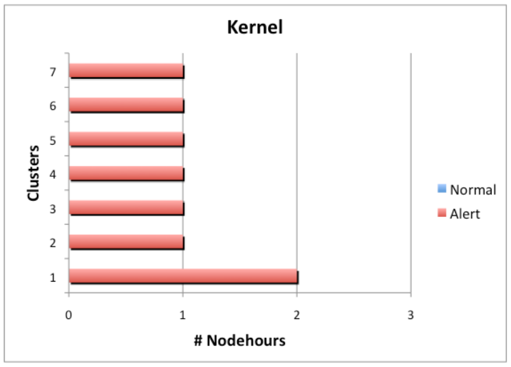


(c) Xen

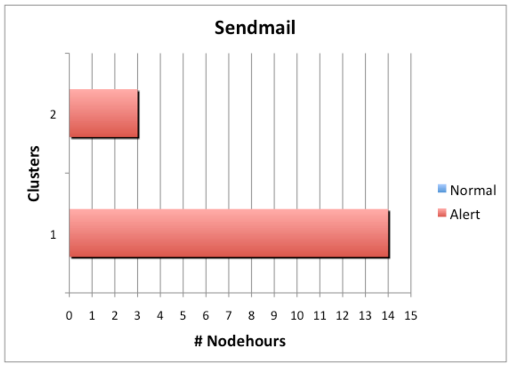
Figure 7.2: Scatter-Plot of nodehours (x-axis) vs. ICSs (y-axis) for the application logs from the cloud log. The plot differentiates between alert nodehours and normal nodehours. Nodehours are sorted based on the ICS values in the plot.



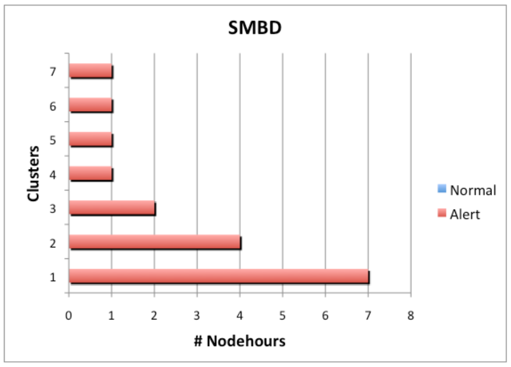
(a) Avahi



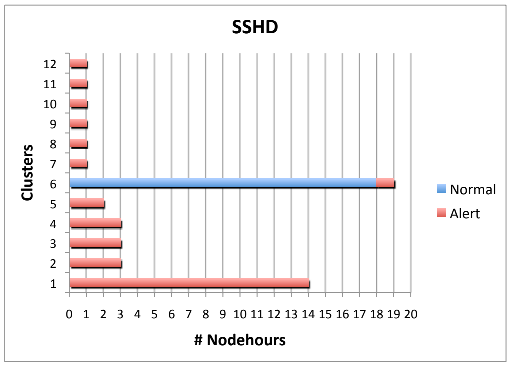
(b) Kernel



(c) Sendmail

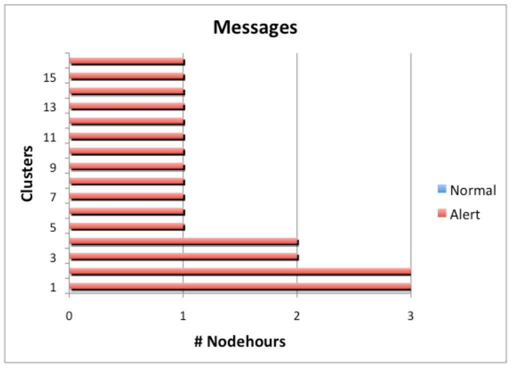


(d) SMBD

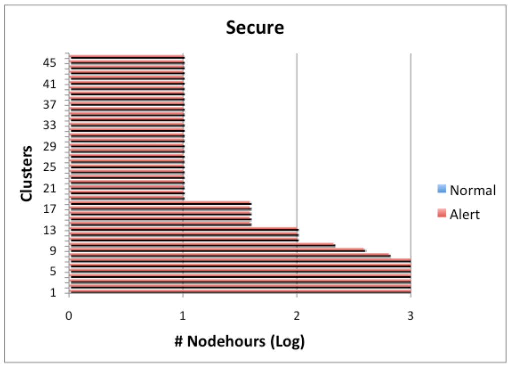


(e) SSHD

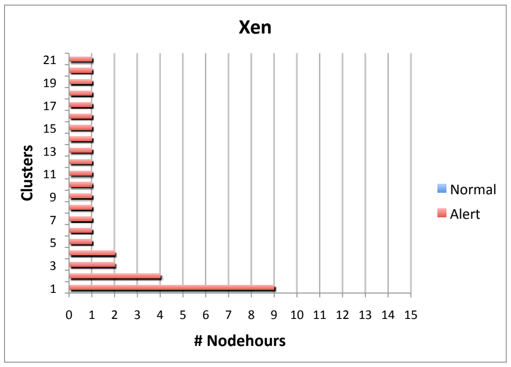
Figure 7.3: Stacked Bar Graphs for the application logs from the distributed log. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). All clusters which contain at least one alert nodehour are shown.



(a) Messages



(b) Secure



(c) Xen

Figure 7.4: Stacked Bar Graphs for the application logs from the cloud log. In each graph, each line on the y-axis represents a nodehour cluster which contains at least one alert nodehour. The colors in each line represent the distribution of alert nodehours (red) to normal nodehours (blue). All clusters which contain at least one alert nodehour are shown.

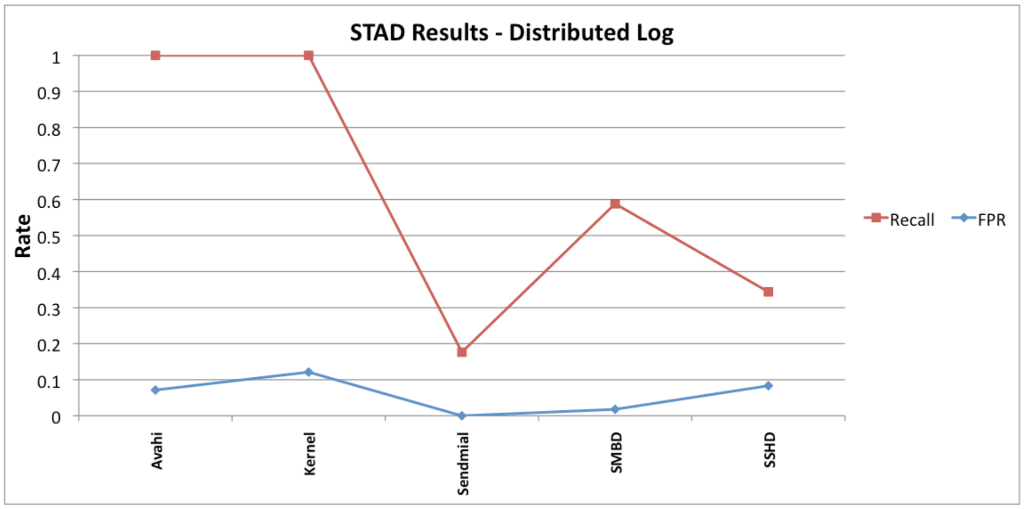


Figure 7.5: Recall and FP rates for STAD on the application categories of the distributed log.

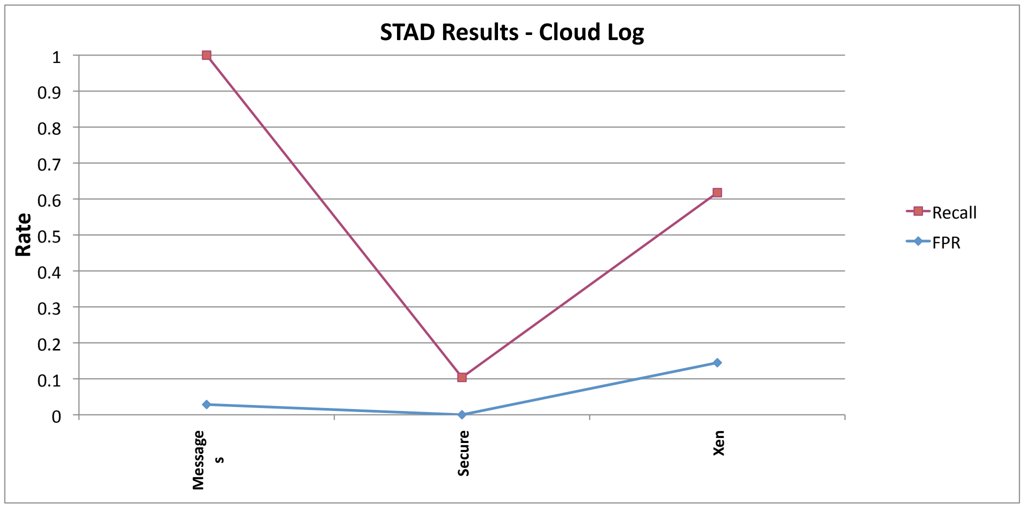
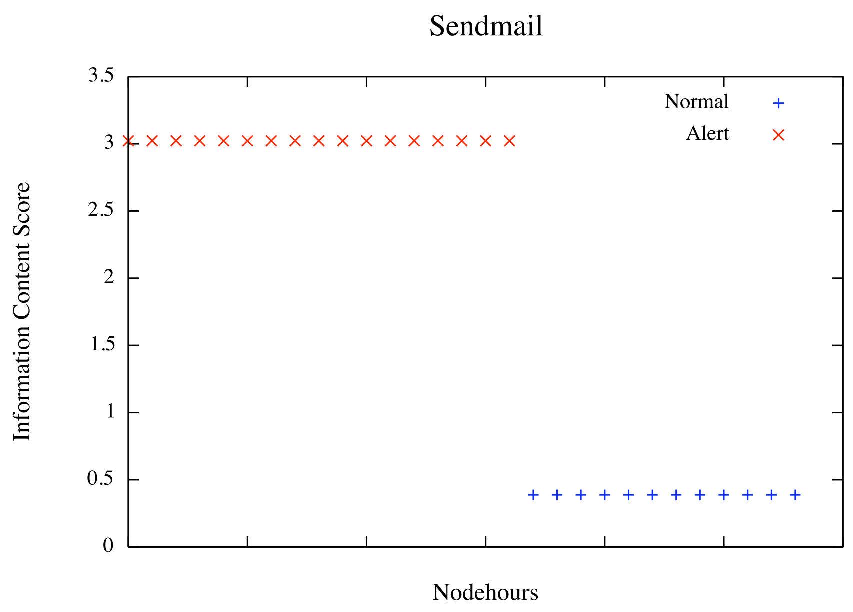
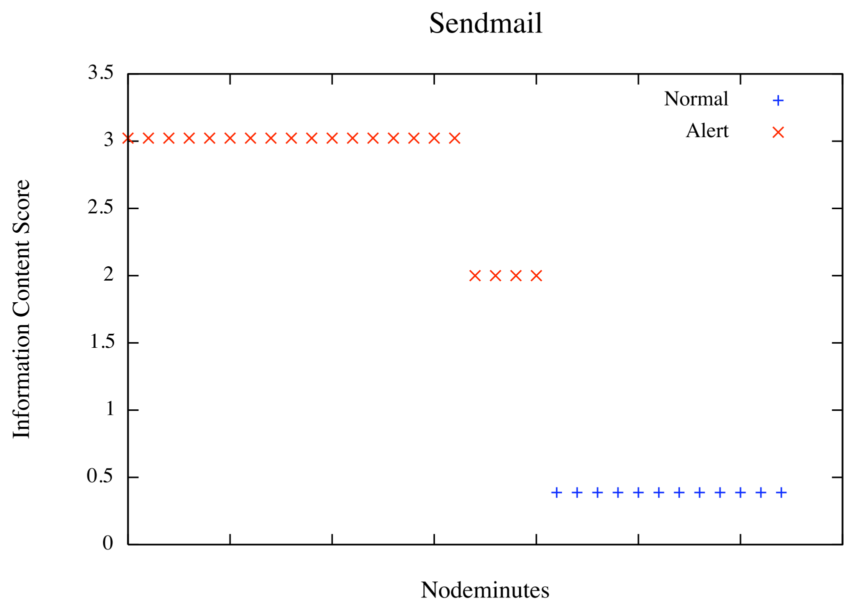


Figure 7.6: Recall and FP rates for STAD on the application categories of the cloud log.

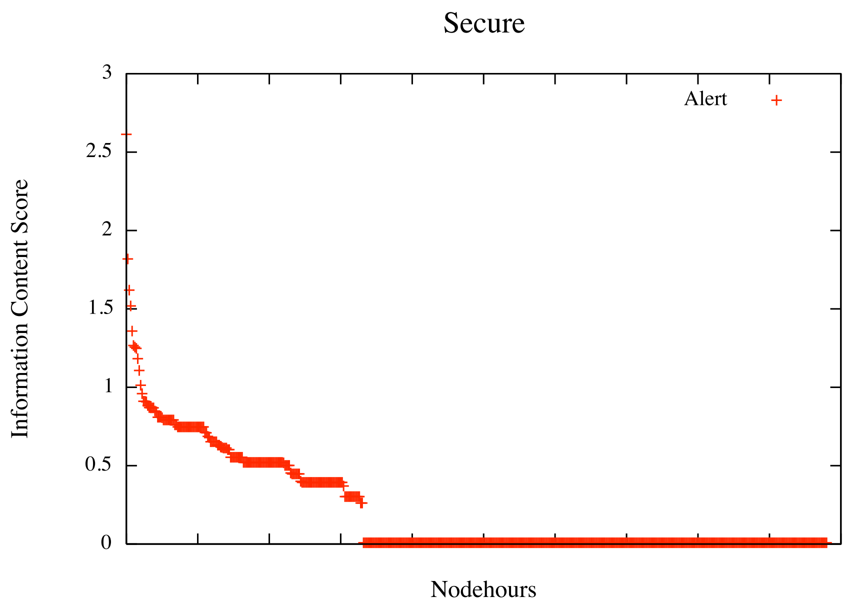


(a) Sendmail - Nodehours

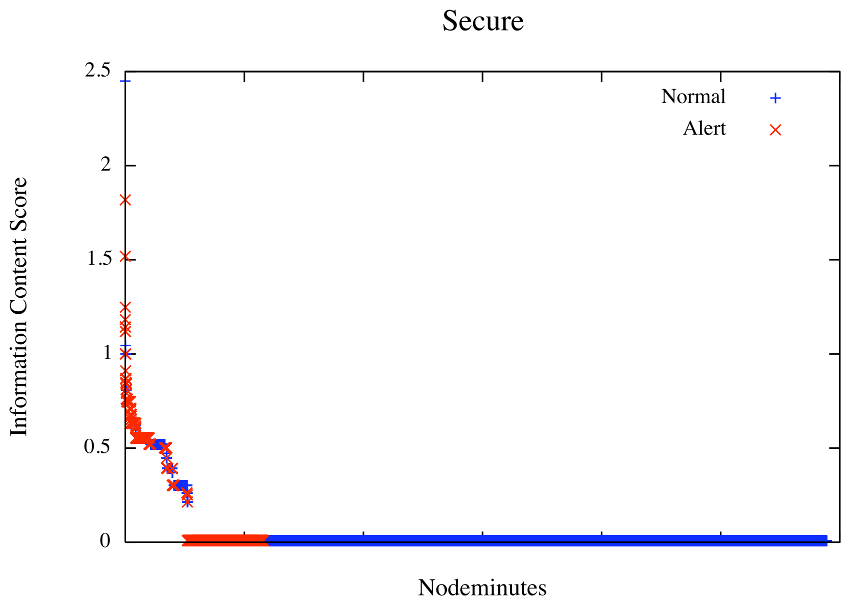


(b) Sendmail -Nodeminutes

Figure 7.7: Scatter-Plot of spatio-temporal partitions (x-axis) vs. ICSs (y-axis) for the Sendmail and Secure application logs. For each application, the temporal granularity level of the plot is "Hours" for the graph on the left while "Minutes" are used for the graph on the right.



(c) Secure - Nodehours



(d) Secure - Nodeminutes

Figure 7.7: Scatter-Plot of spatio-temporal partitions (x-axis) vs. ICSs (y-axis) for the Sendmail and Secure application logs. For each application, the temporal granularity level of the plot is "Hours" for the graph on the left while "Minutes" are used for the graph on the right.

Chapter 8

Putting it All Together: Online Alert Detection through Interactive Learning of Alert Signatures

The ability to discover error conditions automatically with little human input is a feature lacking in most modern computer systems and networks. However, with the ever increasing size and complexity of modern systems, such a feature will become a necessity in the not too distant future; modern computer systems and networks are increasing in size and complexity at an exponential rate. The goal of automatic alert detection is to create systems which are able to identify their own error conditions/states with limited input from a human operator.

The material in this chapter presents the results which demonstrate how the alert detection framework proposed by this thesis would perform on real life production systems. The framework utilizes all of the techniques which have been presented in previous chapters. The use of these techniques is optional. While most of the techniques used are suitable for offline analysis, this framework shows how knowledge extracted from off-line analysis can be used online. Off-line anomaly detection detects the parts of the log which are likely to contain errors (anomalies). Via visualization, human administrators can inspect these anomalies and assign labels to clusters which correlate with error conditions. The system can learn a signature from the confirmed anomalies which it uses to detect future occurrences of the error condition on a production system. The evaluations suggest that the system is able to generate simple and accurate signatures using very little data. This framework contributes to the goal of achieving self-healing systems in autonomic computing.

This framework differs from and improves on previous approaches due to its hybrid nature and its use of techniques which have low computational cost. Previous approaches have utilized either a signature-based [21] or anomaly-based [42] approach but not both. Visualization techniques, if included, are not used interactively and are used only for the visualization of the results of analysis [2, 82]. The inclusion of a signature-based component will help to improve accuracy, while the visualization component helps the system to learn from the expert knowledge of the human administrator.

The material presented in this chapter can be found in this publication [52].

8.1 Methodology

The main idea behind the proposed online alert detection framework is to use the content of historical logs to learn the signatures of previous alert conditions which may have manifested in the log, with the proviso that this be done with minimal human intervention. Once these signatures are learned, they can be stored and applied on production systems to detect future incidents of such alerts as they occur. The proposed system utilizes anomaly detection, signature generation and visualization to achieve this goal. The framework is a hybrid signature- and anomaly-based system.

Signature-based systems work by utilizing models of known and well defined behaviors of interest. They are very accurate but suffer from their inability to detect new behaviors of interest. Anomaly-based systems, on the other hand, require less apriori knowledge of the behaviors the system aims to detect. Instead, they attempt to identify behaviors that differ from the *norm* which enables them to detect new behaviors of interest. However, they tend to be less accurate as legitimate behavior can differ from the *norm*. By utilizing concepts from both signature- and anomaly-based systems, the framework builds on the strengths of both systems, by both being capable of detecting new types of alerts and being as accurate as a signature-based system.

Due to the complexity of the problem, previous work asserts that in such situations human involvement is important to complement the automated system [3]. Thus, the visualization component provides the interactive component of the framework. It allows the human administrator to interact with the proposed system and provide feedback which will improve the models learned by the system over time and keep them up-to-date.

The diagram in Fig. 8.1 provides an overview of the phases of the proposed alert detection framework. The phases of the framework are:

1. Unsupervised anomaly detection through the clustering of system logs and the

separation of normal and anomalous log clusters.

2. Feedback from the human operator in the form of identification and labeling of anomalous clusters.

3. Mining of alert signatures and cluster refinement. The feedback from the administrator is used for learning alert signatures and for cluster refinement

4. Repeat Steps 1 - 3 as required.

The framework does not require the use of any specific mechanisms for carrying out its phases. The choice is left to the discretion of the user.

For the anomaly (alert) detection component of the system, the use of STAD [51] is proposed. The steps of the STAD framework are explained in Section 8.1.1. STAD is described in more detail in Chapter 6. STAD works by clustering the spatio-temporal partitions of the log and separating the clusters into normal and anomalous categories. These categories contain several different types of *normal* and *anomalous* behavior, hence the clusters would require labels i.e. textual description and categorization. These labels can be provided by the human administrator through a visualization system. A system such as LogView [41] could serve this purpose, albeit with a modified hierarchical structure which would include spatio-temporal partitions and the clusters identified by the system. The level of user interaction would be minimal, as the system does not require a complete labeling of all clusters to learn their signatures. A label on a single cluster is enough for the system to learn a signature if need be.

Once these labels are acquired, the system can take all the alert clusters with the same label and generate a signature for them. A frequent itemset mining paradigm is used for signature generation: it is described in more detail in Section 8.1.2. The anomaly detection system continues to improve on its accuracy by using the information from the labels provided by the administrator and the alert signatures it generates to improve its clustering and anomaly detection accuracy. Examples of how this can be done are given in Section 8.1.3.

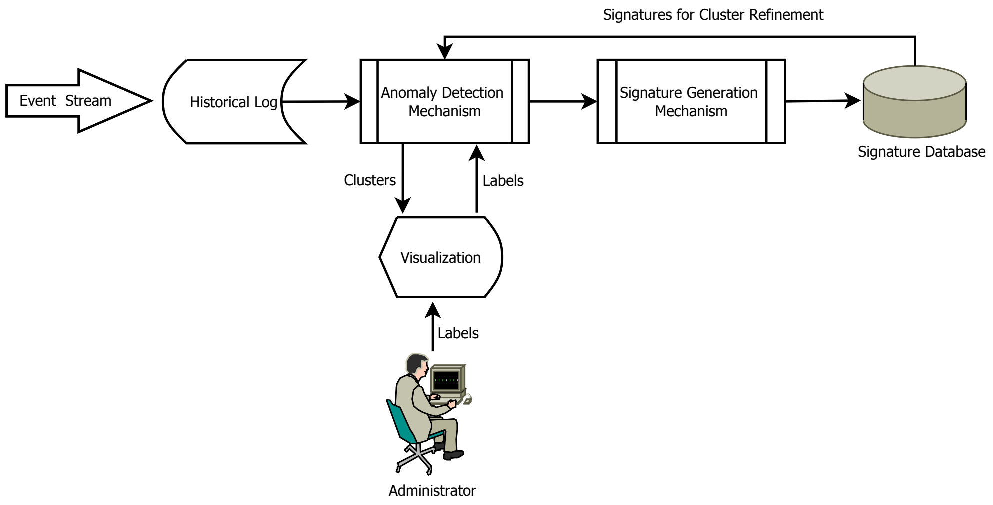

Signatures for Cluster Refinement
Event Stream
Historical Log
Anomaly Detection Mechanism
Signature Generation Mechanism
Signature Database
Clusters
Labels
Visualization
Labels
Administrator

Figure 8.1: Framework for Automatic Signature Generation from System Logs. This figure highlights the phases involved in the proposed framework for automatic alert signature generation using the contents of the system's historical log files.

8.1.1 Anomaly Detection

The steps of the STAD framework are summarized in this section, Fig. 8.2 shows the steps of the STAD anomaly detection mechanism. The techniques used for each step of the framework have been discussed in previous chapters.

- **Message Type Transformation.** This involves the transformation of the natural language description of each line in the log into a unique token which represents the message type that produced the description. Previous work has shown that message type transformation reduces not only the amount of computation required for log analysis but may improve the accuracy of the analysis as well [82, 2, 42].

- **Spatio-Temporal Decomposition.** Spatio-Temporal decomposition involves the partitioning of the contents of the log such that each partition contains all events from a single source over a specified time period. This can be done easily on most logs, using information from their timestamp and the other fields in the event header.

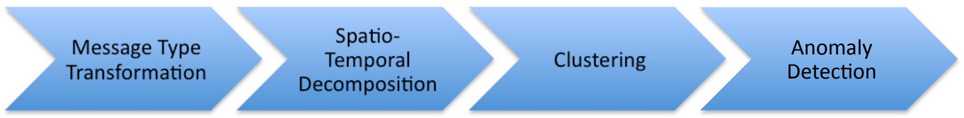


Figure 8.2: Spatio-Temporal Alert Detection (STAD). This figure shows the steps in the STAD alert detection mechanism. The mechanism results in the classification of the contents of log into *normal* and *anomalous* categories.

- **Clustering.** This step requires the grouping of the spatio-temporal partitions such that each group contains partitions which are very similar to each other, while being very dissimilar from the spatio-temporal partitions in other groups. The exact method which is used to achieve this can be chosen freely by the user.
- **Anomaly Detection.** It is assumed that the resulting clusters from the previous step contain either a majority of normal activity or a majority of anomalous (alert) activity. The anomaly detection step serves to separate such clusters. Again, the exact method used is left to the discretion of the user.

8.1.2 Signature Generation

The input to the signature generation mechanism (see Fig. 8.3) consists of the spatio-temporal partitions belonging to the cluster(s) which have been identified by the human administrator as containing activity which relates to a particular alert type. The administrator would have to review only those clusters identified by the system as being anomalous. Subsequently, a cluster pruning step is performed, as described in Algorithm 9. During this step the set of message types reported in each spatio-temporal partition is pruned by computing iteratively the difference between the set of message types in the spatio-temporal partition and the cluster centroids for each of the clusters identified as *normal* by STAD. The method for choosing cluster centroids is described in Chapter 5.

After pruning, frequent itemset mining is performed on the spatio-temporal partitions. Given a set of objects or items (called an item base), S, we define $\mathbf{T}$ as a set of transactions defined over S such that for all $T \in \mathbf{T}$, $T \subseteq S$. A subset (also called an itemset) S' of S is said to be *frequent* if the number of transactions in $\mathbf{T}$ which

Algorithm 9 The cluster pruning pseudo code.

```
Input: The spatio-temporal partitions which need to be pruned, A. The cluster centroids
    for all the spatio-temporal partition clusters identified as normal by STAD, B.
Output: The pruned spatio-temporal partitions
1: for each A as $A$ do
2:   {$A$ will contain a set of message types.}
3:   for each B as $B$ do
4:     {$B$ will also contain a set of message types.}
5:     $A = A \backslash B$ {Compute set difference}
6:   end for
7: end for
```

are supersets of S' exceed a user-specified support threshold and is referred to as a *frequent itemset*. The goal of frequent itemset mining is to find all itemsets which occur in **T** with a minimum support threshold. The apriori algorithm is a classical algorithm for frequent itemset and association rule mining [1] and can be used here. Other frequent itemset mining algorithms such as Eclat [85] and FP-Growth [18] can be used as well.

The apriori algorithm is a level-wise iterative search algorithm. It uses its knowledge of frequent k-itemsets to generate and search the space of $(k+1)$-itemsets. It starts by finding all frequent 1-itemsets. Then it uses the set of frequent 1-itemsets to generate the set of frequent 2-itemsets. It continues this until it generates the set of frequent k-itemsets beyond which no more frequent itemsets with a size larger k can be found. The computation at each level in the iteration involves two steps.

1. **Candidate Generation.** If the set of k-itemsets is C_k and the set of frequent k-itemsets is L_k (i.e. $L_k \subseteq C_k$) then the goal of this step is to generate C_k using L_{k-1}. It does this by performing a join of L_{k-1} on itself, i.e. $L_{k-1} \bowtie L_{k-1}$.

2. **Candidate Pruning.** The goal of this step is to generate L_k from C_k. It does this by removing all elements of C_k which are not frequent in the transaction database.

However, a problem exists with the pruning step: the size of C_k may be prohibitive. The direct approach to pruning C_k would involve determining the count of each

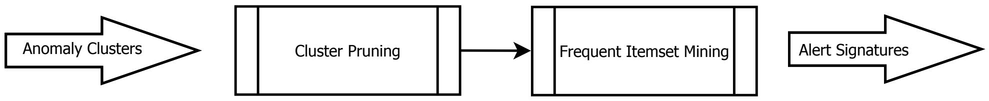


Figure 8.3: Signature Creation Mechanism. This figure shows the steps in the signature creation phase of the framework.

element of C_k in the transaction database, which will require a lot of computation if C_k is large. The apriori algorithm utilizes the apriori property to reduce the amount of computation required for candidate pruning at each level of computation. The apriori property states that all nonempty subsets of a frequent itemset must be frequent as well [17]. The algorithm gets its name from this property. Using this property, it can be stated that a $(k-1)$-itemset which is infrequent cannot be a subset of a frequent k-itemset. Hence, if any subset of size $(k-1)$ of an itemset in C_k is not in L_{k-1}, such an itemset cannot be frequent and can be removed from C_k. This is how the apriori algorithm reduces the amount of computation required for the frequent itemset search.

The frequent itemset mining paradigm can be applied to the problem of alert signature generation by supposing that message types are items and a spatio-temporal partition is a transaction. Hence, the set of message types found in a system log is the item base and the transaction database is the set of spatio-temporal partitions found in the log. It is theorized that for any set of related transactions (spatio-temporal partition cluster) in the transaction database (system log), the set of frequent itemsets which occur in the transaction cluster would be an effective signature for identifying future occurrences of that transaction type. If a transaction cluster is related to an alert type, then the frequent itemsets mined from such a cluster would form a signature for that alert. Given a spatio-temporal partition cluster **C** which contains some sort of alert behavior, the set of frequent itemsets derived from **C** (i.e. C^f) will contain the signature for that alert behavior. The alert signature defined by C^f is triggered if a message type which is contained in any of the itemsets in C^f is found in a spatio-temporal partition.

Once the signatures are generated they can be stored in a database and used to detect future alerts on a production system, Fig. 8.4. This database will be updated

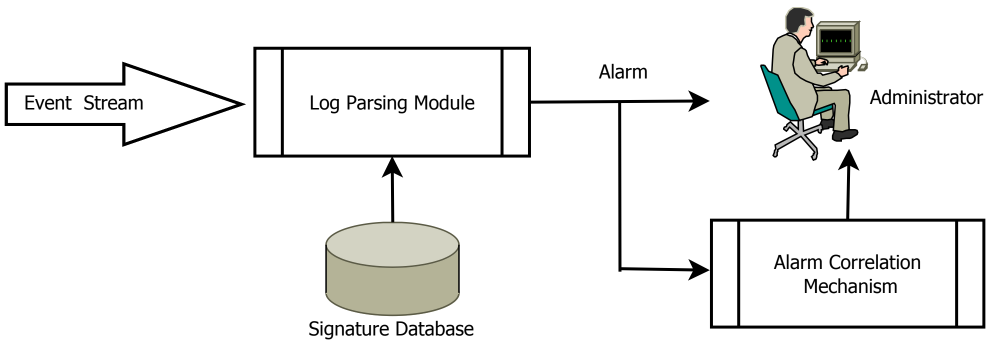


Figure 8.4: Online Alert Detection. This shows how the alert signatures produced by the system can be used for online alert detection on a production system.

as more up-to-date signatures are discovered. The log parsing module would parse the log event stream, searching for any events (or log partitions) which match any of the signatures in the database. If a match is found, an alarm is raised. The alarms can be passed directly to the administrator for action or passed to a downstream module such as an alarm correlation system before being passed to the administrator. The result is a system which can automatically search its own logs for symptoms of failure and notify the administrator. Unlike other systems which do this using signatures which are produced and managed manually, the proposed system produces the signatures itself while relying on the administrator only for class labels.

8.1.3 Visualization, Feedback and Cluster Refinement

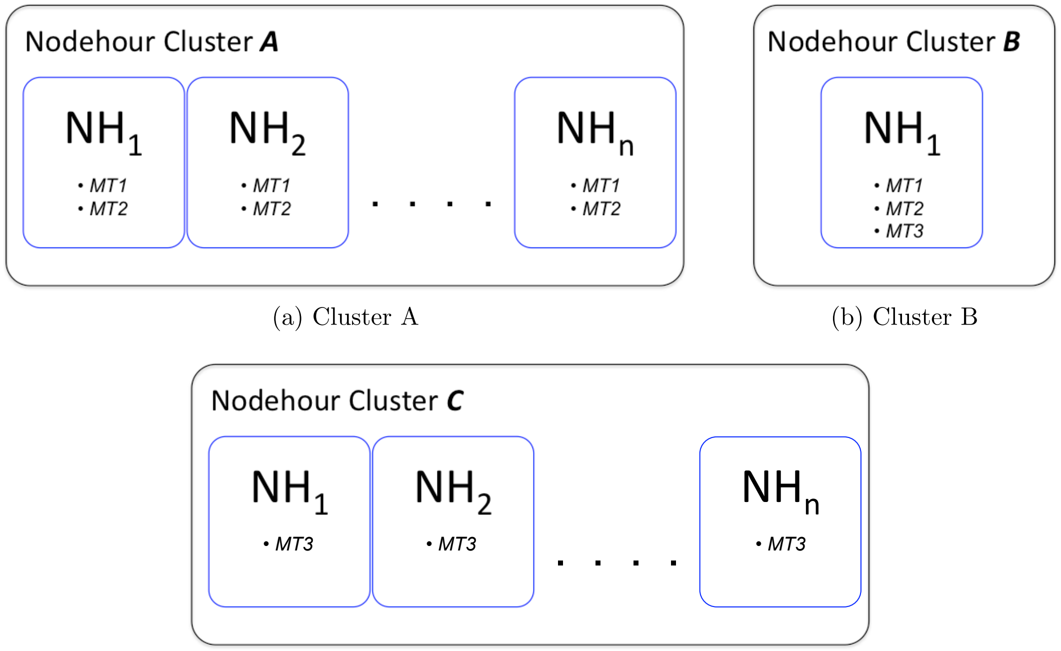
The anomaly detection capability of STAD depends on its ability to cluster the spatio-temporal partitions of a log properly. Nodehours and nodeminutes were chosen as units of spatio-temporal decomposition for the experiments. This decomposition is done arbitrarily: it aims to ensure that the resulting spatio-temporal partitions contain correlated messages types which define a single system state. However, such decomposition may separate correlated message types or may combine correlated message types from different states. The example of Cluster B in Fig. 8.5 is an example of the latter.

In the example in Fig. 8.5, it is assumed that Clusters A, B and C are clusters formed by clustering the nodehours in a log. Furthermore, it is assumed that the anomaly detection process has identified Clusters A and C as normal, while Cluster B has been identified as an anomaly. This classification of Cluster B may be a FP, as its contains a spatio-temporal partition which contains information about two normal states. The spatio-temporal partition is anomalous but the states it defines are not.

The information which the anomaly detection mechanism receives via the feedback loop can be used to resolve this scenario. Let us assume that Cluster A is labelled and identified as a normal cluster through the visualization system by an administrator. The system can use the cluster centroid or a signature learnt through frequent itemset mining as a signature for this normal cluster. In this case, the signature will be $(MT1 \vee MT2)$.

Using this signature in its database, it is now possible for the system to *reason* about the single spatio-temporal partition in Cluster B. It can discover that the spatio-temporal partition contains the complete signature for Cluster A. An *IF* clustering scenario can be created by performing ICC on the spatio-temporal partition without considering the message types which are part of the signature of Cluster A. This action will show that the cluster now belongs to Cluster C. Hence, the system knows that Cluster B is actually a *soft* cluster which belongs partially to Clusters A and C. By knowing that Clusters A and C are normal clusters, it can eliminate Cluster B as an anomalous cluster and reduce the FPs.

Another *reasoning* scenario could occur if, as in the example, the system discovers that a spatio-temporal partition contains complete definitions of system states from other clusters: it can attempt to split the spatio-temporal partition into smaller units. In the single spatio-temporal partition in Cluster B, if two non-overlapping time units, t_1 and t_2 , can be found such that $t_1 + t_2 = 1hr$ (if the spatio-temporal partition is a nodehour) and all instances of $MT1$ and $MT2$ occur in t_1, while all instances of $MT3$ occur in t_2, it is possible to decompose the spatio-temporal partition into two new spatio-temporal units, i.e. a node-t_1 and a node-t_2 as in shown in Fig. 8.6. These resulting spatio-temporal units can be placed in Cluster A and Cluster B respectively, eliminating Cluster B altogether.



(a) Cluster A (b) Cluster B

(c) Cluster C

Figure 8.5: The figure above shows three examples of spatio-temporal partition clusters which maybe formed during analysis. For this example, let us assume that Clusters A and C, have been identified correctly as *normal* while Cluster B has been wrongfully identified as *anomalous* due to the fact that it contains only one spatio-temporal partition.

8.2 Complexity

For the proposed framework to be usable in an online environment, its computational cost (time complexity) must allow quick processing of the content of very large logs, which is the trend in today's network systems [68].

At the anomaly detection phase, the costly operations are message type transformation and clustering. The message type transformation is a linear time operation with respect to the number of events in the logs [43]. However, this is a preprocessing step which can be performed in advance. The ICC requires the calculation of a number of entropy-based values. Let W represent the number of terms in the log, C the number of reporting sources and H the number of spatio-temporal partitions. ICC comes after calculations are performed on two matrices of size $W \times C$ and $W \times H$. The time complexity is of the order of $W \times C$ and $W \times H$. However, previous work

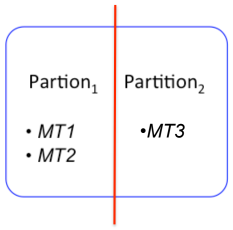


Figure 8.6: The figure shows the single spatio-temporal partition from Cluster B in Fig. 8.5. It depicts a possible scenario of breaking up the spatio-temporal partition into smaller and temporally uneven spatio-temporal units.

has demonstrated that the use of message types to represent the terms in the log, i.e W, significantly reduces the dimensionality of the problem by an order of about a hundred [42]. The actual clustering step which comes after these calculations is a linear time computation with respect to H.

Let n represent the number of items in an item base. The time and space complexity for mining the set of frequent itemsets from a transaction database defined over the item base is theoretically of the order of $\sum_{k=1}^{n}\binom{n}{k}$, i.e. the number of possible itemsets which can be generated from the items in the item base. However, in practice, the size of the largest candidate itemset is more likely bounded by the size of the largest transaction in the transaction database, which is usually a lot less than the number of items in the item base. The use of the classical apriori algorithm helps to reduce this complexity further by reducing the number of candidates itemsets which need to be generated. In addition, there are several published approaches using sampling, partitioning, transaction reduction etc., which can improve the efficiency of the process even further[17].

This discussion forms the basis on which this thesis argues that the methods used all scale gracefully in the face of large logs.

8.3 Evaluations

The experiments involved simulating the alert generation mechanism of the framework using historical log data and then using the signatures to detect alerts in spatio-temporal partitions as they occurred. The goal was to measure the detection accuracy of the generated signatures. The following assumptions were made.

- The signatures are static, i.e. once they are created, they do not change.
- The anomaly detection portion of the framework works perfectly, i.e. it is able to separate *normal* and *anomalous* clusters perfectly.
- The administrator is able to label all alert clusters shown to him/her by the system accurately. As these labels are the basis on which the system decides from what alert clusters to mine signatures, the accuracy of this labeling will have an impact on the result.

The visualization component of the system allows the human operator to interact with the system and provide labels for signature generation and cluster refinement. Since cluster refinement was not performed in these experiments and it is assumed that all labels have been provided accurately, the visualization phase was not evaluated in this thesis.

Each experiment is done for only a single alert type using datasets derived from the HPC, Distributed and Cloud logs mentioned in previous chapters. Statistics of these datasets are provided in Tables 4.7, 7.2 and 7.3 respectively.

Each log dataset is split into five separate training and testing pairs for each alert type (see Tables 4.7, 7.2 and 7.3). These training and testing pairs are not of equal size. The split point for the training file is determined by the time at which 10%, 20%, 30%, 40% and 50% of all spatio-temporal partitions of the alert type has occurred. All log events occurring after each of these points are used to test the signatures found. Each run involved the execution of the proposed framework on a test file, which was assumed to be a historical log, to learn the signature for an alert type. The generated signature(s) were applied to the test set to detect spatio-temporal partitions which contained the alert type after which the number of TPs, FPs,TNs and FNs were determined. These values were used to calculate the detection rate (recall) and FPR as defined in Eq. 3.4 and Eq. 6.4 respectively. In summary, as a result of this setup, approximately 1,120 experiments were carried out.

An efficient open-source implementation of the apriori algorithm [1] was utilized in all the runs [5]. The support threshold was set to 50% for all runs. It is not uncommon for frequent itemset mining to generate a large number of itemsets which are

[1]Downloadable from *http://www.borgelt.net/apriori.html*

sometimes redundant. For this reason, only *closed* frequent itemsets were generated. Frequent itemsets which have no superset with the same support value are referred to as *closed*.

8.4 Results

This section present results on the detection accuracy of the signatures and the nature of the signatures produced by the proposed system.

The FPRs achieved during the experiments show very little variation. The FPR achieved was approximately 0% for all runs. These FPRs are operationally acceptable and show that the goal of converting detected anomalies into signatures as a means of reducing false positives was achieved.

An average DR of 88%, 84% and 83% was achieved on the HPC, Distributed and Cloud logs, respectively. The distribution of these DRs is shown in Fig. 8.7. However, it is stated that a DR close to 100% can be expected for most of the signatures produced for the HPC logs: Fig. 8.7(a) shows the DR distribution for the runs for each log file in the HPC category. It can be seen that the average DR would be much higher if the few *outliers* are not considered. Investigations show that some of the outlier DR values are due to the quality of the clusters produced or due to the distribution of signature(s) across time. In cases where the alert state is not the majority behavior in a cluster, the signature learnt would not relate to the alert and hence would produce a DR close to zero. This situation can be mitigated by reducing the support threshold used in signature generation. In cases in which signature(s) for an alert are not distributed evenly in time, it is possible to only learn the signature(s) for an alert partially. Leading to less than perfect detection, this situation is pronounced particularly in datasets *4,9,11,13* in Fig. 8.7(a). In a real life implementation the signatures would not be static, as is the case in the experiments. Having dynamic signatures, as would be the case in an online implementation, will allow the proposed system to learn new signatures for an alert and improve its detection accuracy. In Fig. 8.7(b) there are no DR results for the *Avahi* log due to the fact that there is only one alert type for this log and only one cluster formed at any point in time. Hence there are no alerts to detect after learning the signature. However, the signatures learnt for the *Avahi* file could be said to be effective as they did not generate any

Table 8.1: ANOVA Test Summary

Treatment	**F**	**P-Value**	**F crit**
FPR(HPC)	0.874	0.485	2.525
DR(HPC)	0.162	0.957	2.531
FPR(Distributed)	65535	NA	2.866
DR(Distributed)	0.311	0.866	3.055
FPR(Cloud)	65535	NA	3.478
DR(Cloud)	0.068	0.990	3.478

FPs. Though there were no alerts to detect, the *Avahi* alert signatures were tested anyway and having no FPs implies that the proposed framework did not make any errors in misclassifying normal behavior as alert behavior.

The graphs in Fig. 8.8 show how the average DR changes as the size of the training file is changed. The results show that while an increase in DR performance is noticed for 3 log files, i.e. *Bgl-Comp, Tbird-Other* and *Liberty-Other*, as the size of the training file is increased, there seems to be no correlation between the size of the training file and the DR performance. This is confirmed by an ANOVA test performed at 5% significance, which is summarized in Table 8.1. Detailed ANOVA results are provided in Appendix D. The ANOVA results show no statistically significant difference between the FPR and DRs achieved when the training file size is changed. These results suggest that alert signatures can be learnt when as little as 10% of the exemplars for an alert type are present in the log.

The nature of the signatures produced are discussed using three factors: their length, their support value and the number of signatures generated for each alert type. An alert signature, as defined by this research, consists of a set of message types which can be used disjunctively or conjunctively to detect an alert condition in a log file partition. Therefore, the *length* of a signature is the number of message types which define it. This thesis assumes that shorter signatures are not only simpler but better defined, i.e. compact, so shorter signatures are deemed to be preferable than longer ones. The graphs in Fig. 8.9 show the length distribution of the signatures for each of the HPC, Distributed and Cloud logs. The results show an overall median length of two and mean length of six for the HPC logs, median length of six and mean length of ten for the Distributed logs and a median length of four and mean length of twenty-seven for the Cloud logs. Since the minimum signature length is one, these results show that the signatures produced by the proposed system are relatively simple and compact. High mean lengths are due to the effect of outliers.

Signatures are generated as frequent itemsets from clusters of spatio-temporal partitions, therefore each signature has a support rate and since more than one frequent itemset can be mined, it is possible to have more than one signature for an alert type. While it can be debated, it is safe to say that signatures with higher support rates are likely to be better detectors than those with lower support rates. Therefore, this work reports on the support rates for the signatures produced during the experiments. The graph in Fig. 8.10 shows the support rate distribution for the signatures. The results show an overall median support rate of 100% and mean support rate of 94.45% for the HPC logs, an overall median support rate of 100% and mean support rate of 95.71% for the Distributed logs and an overall median support rate of 100% and mean support rate of 82.96% for the Cloud logs. These support rates are very high and attest to the quality of the signatures produced. The results are not surprising given the high internal cohesion rates reported from the assessment of the clusters produced using ICC [50]. Generating a few (ideally one) signature(s) for each alert type is desirable. The data in Tables 8.2, 8.3 and 8.4 report on the median and maximum number of signatures created for each alert type. The median number of signatures was the minimum (1) for all the log files except *Liberty-Compute*, *Liberty-Admin*, *Spirit-Comp*, *Sendmail* and *Messages*.

It is important to mention that not all of the experiments were able to produce useful signatures: of the 1,120 experiments performed, useful signatures were not produced directly in 137 (12%) of them. After analysis, it was discovered that the inability of the framework to produce signatures was due to one of two reasons:

- Very high support value.
- Over-generalized message type descriptions.

The minimum support value set for all experiments was 50%. In 32 experiments signatures could not be produced because this support value was too high. If these experiments were run with lower support values, useful signatures would have been produced. This is a parameter which can be tuned in an actual implementation of this

Table 8.2: Number of Signatures Per Alert Type

	# Alert Types	Median	Max
BGL-Compute	18	1.0	6.0
BGL-IO	17	1.0	2.0
BGL-Link	5	1.0	2.0
BGL-Other	7	1.0	2.0
Liberty-Compute	17	2.0	5.0
Liberty-Admin	3	2.0	4.0
Liberty-Other	7	1.0	9.0
Spirit-Compute	29	2.0	33.0
Spirit-Admin	3	1.0	2.0
Spirit-Other	11	1.0	3.0
Tbird-Compute	31	1.0	36.0
Tbird-Admin	7	1.0	5.0
Tbird-Other	10	1.0	2.0

Table 8.3: Number of Signatures Per Alert Type

	# Alert Types	Median	Max
Avahi	1	1	1
Kernel	3	1	2
Sendmail	1	2	2
SMBD	2	1	2
SSHD	2	1	2

framework and is not a difficult problem to overcome. In 115 runs, the system ended up having no message types in all spatio-temporal partitions after the cluster pruning step. This is due to the fact that the message types which defined the alerts are sub-types of the message types that were extracted automatically from the logs. This implied that the message types appear in both *normal* and *anomalous* clusters. Hence they end up being deleted in the cluster pruning step. For example, the *APPALOC* alert type found in the *BGL-IO* log is defined by log events which contain the message type *"ciod: Error creating node map from file * Cannot allocate memory."* However the message type extracted from the log was *"ciod: Error creating node map from file * * * *"*, so not all instances of this message are defined as *alerts* in the log. Hence the *"ciod: Error creating node map from file * * * *"* message type appears in some

Table 8.4: Number of Signatures Per Alert Type

	# Alert Types	Median	Max
Messages	3	3	8
Secure	2	1	1
Xen	3	1	2

normal clusters and is pruned whenever it appears in an alert cluster.

This example of an over-generalized message type highlights the importance of message type extraction as the foundation for this framework. The message type extraction process is able to find a *meaningful* message type and though the message type is *meaningful*, it is unable to distinguish the *alert* due to its being over-generalized. Unfortunately, even for a human administrator, the *right* level of abstraction to use in such situations is difficult to guess without prior in-depth knowledge of the alerts beforehand. In such situations, performing the message type extraction process again on only the log events found in the cluster(s) would yield the accurate message type required to distinguish the *alert*. Without the log events which are considered *normal* in those clusters, the message types extracted would be produced at the right level of abstraction and can be used as signatures for the alerts without generating false positives. It is possible for the system at this stage to look for other discriminating features which might differentiate a *normal* occurrence of a message type from an *anomalous* one, using information from other fields in the event log. For example, the *"data storage interrupt"* message type extracted from the *BGL-Comp* log defines the *KERNSTOR* alert type but could be found in *normal* clusters as well. However, *alert* instances of this message type appear with the term *FATAL* usually in the severity field.

8.5 Discussion

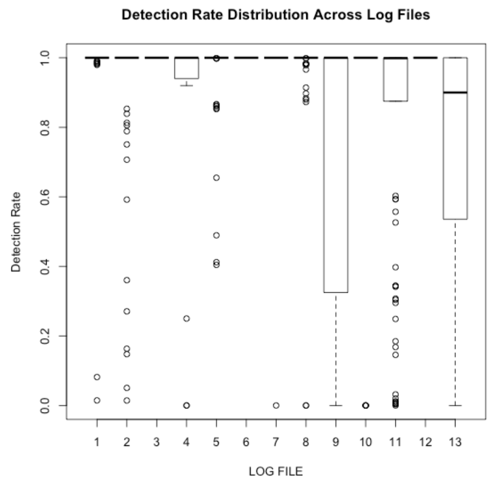
This chapter has presented evaluations of the hybrid alert detection framework for alert detection in system logs as proposed by this thesis. The proposed framework combines the advantages of anomaly-based and signature-based detection. By including an interactive visualization, the framework hopes to provide a window by which human administrators can provide feedback to the system. The system uses this feedback to generate signatures and improve on its anomaly detection capability over time.

The evaluations suggest that effective signatures can be learnt with minimal amounts of data. The signatures learnt are not overly complex and since they are composed of sets of message types, they are human readable. The signatures are found to be relatively accurate. They are able to achieve a DR $> 83\%$ on average, while maintaining an operationally acceptable FPR of approximately 0%. The best case was 100% detection with no false positives. It is possible that higher DRs are achievable by relaxing the tightness of clusters and skipping cluster pruning. However, this will come at the cost of higher FPRs. It will be interesting to investigate this tradeoff in future work.

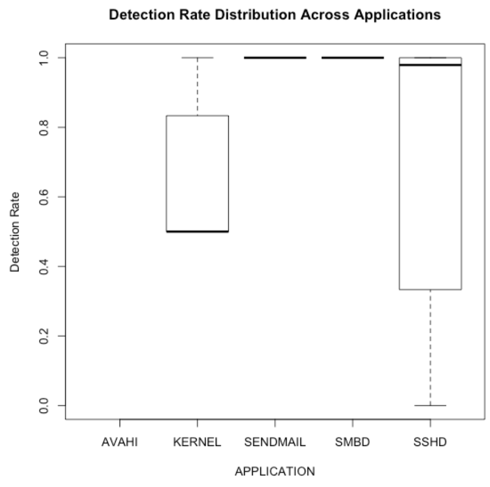
The reduction in the DR for the log data from the distributed and cloud computing infrastructures when compared to the HPC data is most likely due to the difference in the way similarity was defined for the files. Albeit, it is likely that the difference is only related to differences in the data. The level of accuracy attained demonstrates that the goal of achieving high accuracy by introducing a signature-based component into the framework is possible. Achieving high DRs with no FPs is dependent on accurate message type extraction and clustering. The results attest to the importance of message type extraction accuracy as a foundation of the framework.

The signatures created by the system can be created in real-time and can be brought online immediately. This property ensures that the system can be used on production systems.

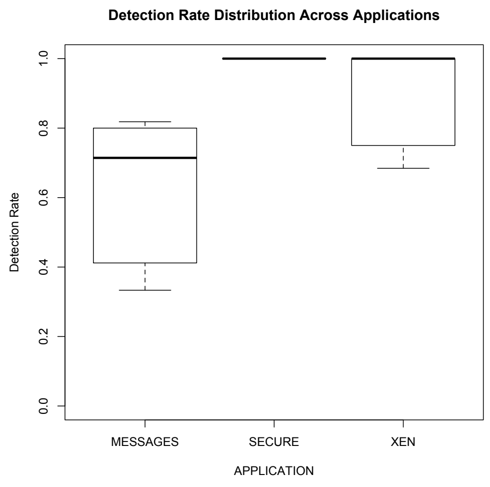
The framework provides a lot of flexibility in the choice of techniques used during each of its phases. None of the techniques used in the evaluation is tied to the framework: all can be changed with user discretion. While nodehours and nodeminutes were used during the evaluations, it is possible to use spatio-temporal partitions with any level of granularity. This is useful especially for real-time alert detection when quick discovery is important. Once learned, signatures can be applied to a spatio-temporal partition at any desired level of granularity.



(a) HPC

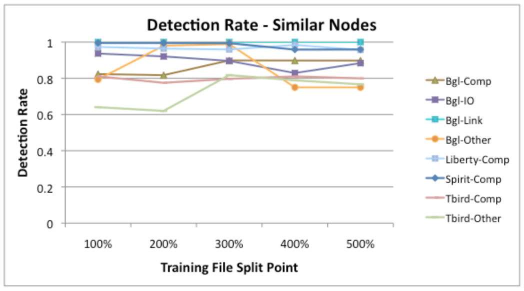


(b) Distributed

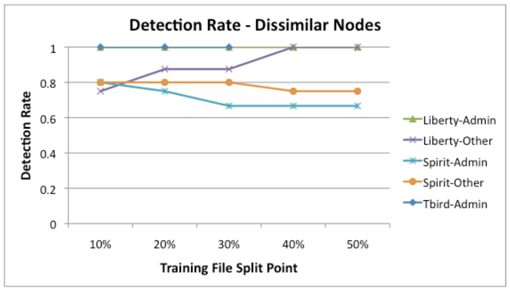


(c) Cloud

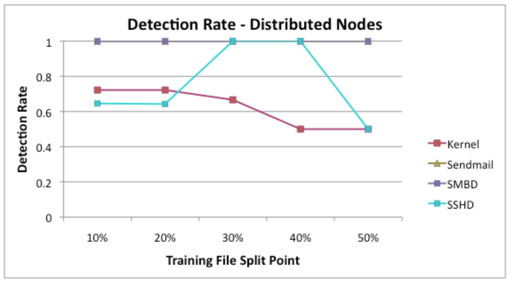
Figure 8.7: Detection rates for the different HPC, Distributed and Cloud logs. For the HPC log, the files are numbered in the same order as shown in Table 4.7



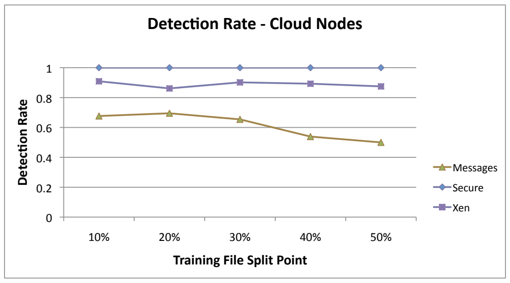
(a) HPC Similar Nodes



(b) HPC Dissimilar Nodes

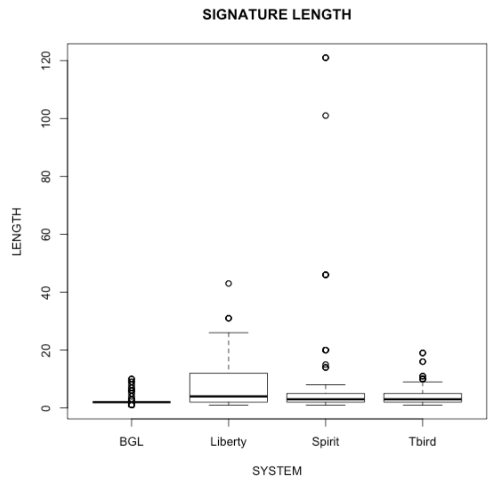


(c) Distributed

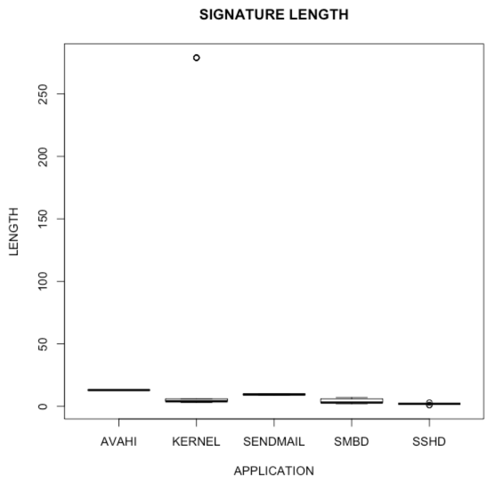


(d) Cloud

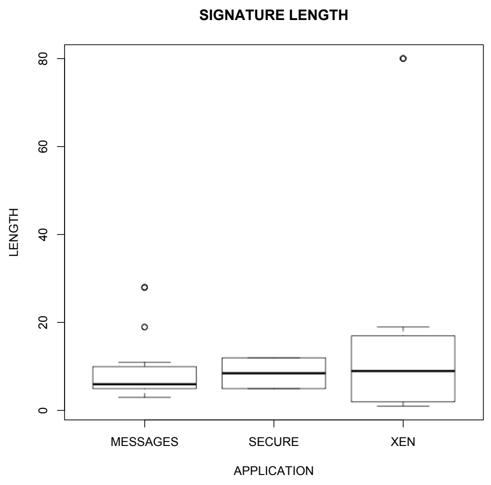
Figure 8.8: Detection rates for the different HPC logs, showing variations in performance for different sizes of training files



(a) HPC

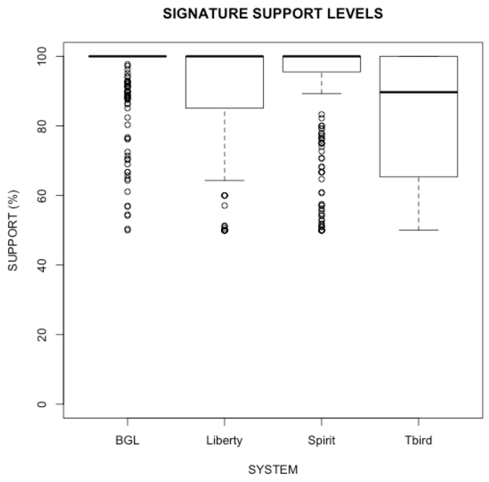


(b) Distributed

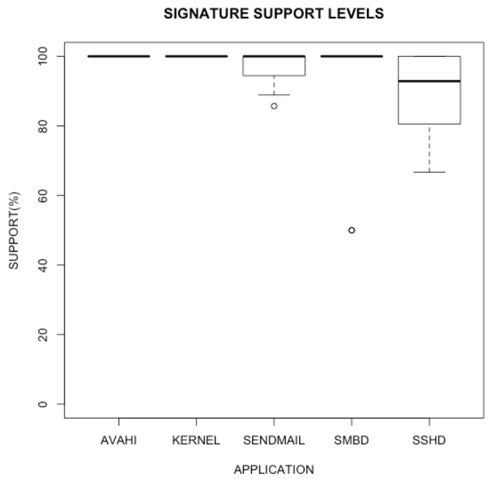


(c) Cloud

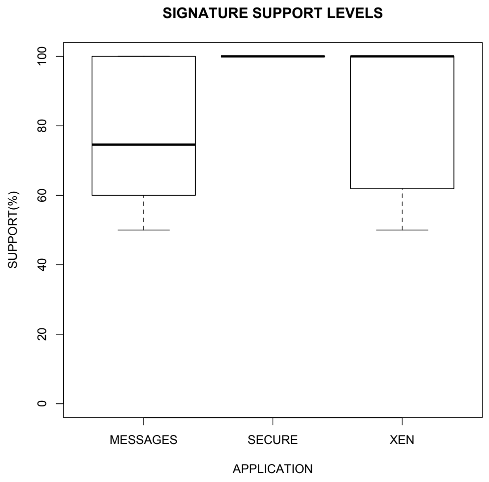
Figure 8.9: Boxplots showing distributions for signature length, i.e. the number of message types which define a signature.



(a) HPC



(b) Distributed



(c) Cloud

Figure 8.10: Boxplots showing distributions for signature support levels

Chapter 9

Conclusion

Autonomic computing can be described as the goal of building self-managing computer systems. The need for self-managing computer systems has become more glaring with the ever increasing size and complexity of today's computer systems. At the heart of autonomic computing are four objectives which are sometimes referred to as various self- properties: e.g. self-configuration, self-optimization, self-healing and self-protection. To enable the autonomic system to meet these objectives it must also posses at least one of the following attributes as well: self-awareness, self-situation, self-monitoring and self-adjustment.

As systems continue to grow in size and complexity, the time and knowledge required for fault resolution must increase in step. Having systems which are capable of self-healing to some degree will help to mitigate this problem. A system which is capable of self-healing is one that is capable of detecting, diagnosing and recovering from its fault conditions with minimal human intervention. The first step in this process is alert detection. Alert detection involves the discovery of the symptoms of the fault condition on the system. Several sources of information are available for the detection of fault symptoms. System logs stand out due to the fact that they play a crucial role in manual fault resolution. Hence, it makes sense to utilize them in automatic alert detection.

This thesis aims to develop a hybrid interactive learning framework for alert detection in system logs. The framework can be placed firmly within the context of autonomic computing, as highlighted in Table 9.1, which is repeated here from Chapter 1. From the foregoing, it can be seen that this goal has been achieved to a significant degree.

Table 9.1: Placing the framework within the context of autonomic computing

Property	**Category**
Objective	Self-Healing, Self-Protection
Attribute	Self-Awareness, Self-Monitoring
Design Approach	External
Framework Type	Technique-Based, Injection of Autonomicity into Non-Autonomous Systems
Autonomicity Level	3-Predictive

The framework is interactive as it includes a visualization component through which feedback can be given. It is hybrid because it utilizes a mix of anomaly detection and signature generation. These components of the system work through a bottom-up approach to detect alerts in the log. The approach involves several techniques which are used for message type extraction, message abstraction, entropy-based analysis, clustering, identification of anomalous clusters and signature generation. Most of the techniques used here are novel contributions of this thesis.

Evaluation of the proposed framework involved piecemeal evaluations of the various techniques used in the framework. These evaluations were discussed in previous chapters. The final evaluation of the framework involved testing the accuracy of the signature generation phase using data from HPC systems, distributed systems and cloud systems. This set covers a wide range of the computer systems used today in the real world. The signatures which were generated by the proposed framework automatically were found to be relatively accurate, able to achieve a DR $> 83\%$ on average, while maintaining an operationally acceptable FPR of ~0%. This result implies that if the cluster(s) which define an alert can be isolated successfully by the anomaly detection mechanism, the system can generate a signature automatically which can detect $> 83\%$ of future occurrences of the alert with negligible FPs. Achieving a very low FP is very important here; a significant FP rate can cause the alarms raised by such a system to be ignored by human administrators.

To the best of my knowledge, this work is the first to propose and provide evaluation results for a hybrid, visualization, anomaly detection and signature generation system for alert detection in logs. Hence, there is no baseline with which to compare it. However, I believe that the average performance, $> 83\%$ DR, ~0% FPR, of the proposed system is a very promising starting point given that the system is automatic and data driven.

9.1 Contributions

The contributions of this thesis will be discussed with respect to the objectives outlined in Section 1.3 of Chapter 1. For each of the six objectives this section will detail the contributions of the thesis which contribute to the meeting of the objective. The contributions follow below.

1. **Minimum Apriori Information.** The alert detection framework proposed by this thesis assumes very little about the infrastructure on which it will run. Indeed, the only significant piece of apriori information which the system needs to be aware of are the similarity categories used for anomaly detection. Automating this step through an analysis of the message types produced by different *sources* can be an interesting direction for future work.

   Also, the framework can detect alerts successfully without semantic analysis. This enhances the platform portability of the framework. The use of semantic analysis would require a taxonomy which counts as significant apriori knowledge.

2. **Unstructured Data.** As detailed in Chapter 1 and Chapter 3, the unstructured nature of log data is a major hinderance to the automatic analysis of log data. Message types can help with the mitigation of this problem. Log data which has been preprocessed through MTT is easier to analyze, produces more accurate models and requires less computation [42, 21]. Unfortunately, message types are not always known, hence the need for message type extraction.

   Message type extraction, like most processes which involve textual data, can be computationally expensive and the accuracy of extraction cannot be guaranteed. The IPLoM message type extraction algorithm is an important technical contribution of this thesis. IPLoM does not require apriori knowledge of the domain from which the log data emanates, as is required by some previous approaches which discover message types by parsing source code or searching the log data for patterns of well known variable tokens such as IP addresses.

It works in linear time, thus reducing the computational expense associated with message type extraction. Furthermore, it is more accurate than previous approaches.

IPLoM can be made a lot faster through parallelization: its decomposition of the message type extraction problem makes it a natural candidate for parallel implementation. While IPLoM's output is accurate enough for use as demonstrated by this thesis, its accuracy can be improved upon by applying more complex mining schemes on its output. IPLoM's output reduces data size greatly, so the use of computationally expensive schemes becomes tractable. Using IPLoM in a production environment would involve running it against historical log data at periodic intervals to discover previously unknown message types. This approach will suffice, as it is known that messages types reported in log data remain relatively static over time [2].

3. **Interactive Learning.** Interactive learning can enhance the results of the computer-based analysis of complex systems to a great extent [3]. One way of achieving interactive learning is through the use of visualization. This thesis proposes a scheme which allows log data to be visualized as a hierarchy of clusters using treemaps [65]. This scheme is implemented in a prototype interactive visualization tool called LogView.

   The scheme allows for flexibility in the hierarchy definition which makes it suitable for STAD. STAD detects anomalous clusters of log partitions. The interactive nature of the LogView prototype ensures that administrators can provide feedback to the anomaly detection mechanism. This allows the framework developed in this thesis to be implemented as an interactive learning framework, which is an improvement on previous work.

4. **Hybrid Detection.** Frameworks for alert detection in previous work have followed either a Signature-based or an Anomaly-based approach. There are pros and cons for each approach, a hybrid approach would however be able to leverage on the advantages of both approaches. The alert detection framework proposed by this thesis improves on previous work by using a hybrid approach.

   STAD is an anomaly-based detection framework. Like other anomaly-based approaches it is able to detect novel activity which comes with a risk of having a relatively high FPR. The proposed framework combines STAD with a signature-based approach based on frequent itemset mining. The signature generation system generates signatures from identified anomalies. Based on the evaluations, the signatures are accurate, as they generate virtually no FPs. In this way the proposed framework takes advantage of both approaches.

5. **Self-Awareness.** Self-awareness is the attribute of an autonomic system which allows it to be informed about its internal state. Previous research efforts in self-healing using system logs have focussed primarily on the objective of self-healing while ignoring the need for attributes such as self-awareness, which a system must possess before it can be capable of true self-management [13].

   This thesis makes a contribution to enhancing self-awareness through the mining of system logs by correlating events which can be found by decomposing the log spatio-temporally. Correlated events in the log define different states of a system, however discovery can be computationally expensive. Unique to this thesis, this property allows for the timely discovery of an accurate initial set of system states which can form the basis for further analysis.

6. **Computational Cost.**

   Due to the ever increasing size of log data on modern systems, an automatic log analysis framework needs to scale gracefully in the face of large data. If it cannot then it is of no practical use. The framework proposed by this thesis achieves this by utilizing several novel techniques which are linear or pseudo-linear in time and memory cost. Some of these techniques include IPLoM, the *NIUniq* equation for assigning information content scores to log partitions (see Eq. 5.2), ICC and the rule-based detection scheme of STAD.

9.2 Future Research Directions

Alerts manifest themselves in various ways which is why this thesis argues in Chapter 2 that there are two major ways in which alerts manifest in log data. Type-II alerts are the most varied. While this thesis has developed an accurate, robust and flexible framework for alert detection, the large variation in alert behavior may prove to be a problem. For the framework to detect a context sensitive alert successfully, the model most be built using the right similarity *dimensions* which will likely expose the context of the alert. This fact makes the problem of alert detection a difficult one and prevents the development of a framework which can detect all alert types accurately without user input.

For instance, the proposed framework relies on *NIUniq* for ICS assignment. In [48], it is demonstrated that Nodeinfo is better at detecting *bursty* alerts. *Bursty* alerts are Type-II alerts. The *Bytes* detection mechanism [48], though relatively simpler than both Nodeinfo and NIUniq, has been demonstrated to detect *bursty* alert signatures effectively. However, it does not generalize as well as NIUniq, thus it is unable to detect any of the other alert types.

Based on this observation, it is suggested that the future direction of alert detection mechanisms in system logs should consist of not one (probably complex) mechanism capable of detecting all types of alerts, but several (probably simple) detection mechanisms which co-operate. Consequently if the proposed framework is found to be unable to detect some *bursty* signatures, it makes sense to use it in conjunction with *Bytes*, rather than adding more complexity to the proposed framework to allow it to detect all *bursty* signatures. This ensemble learning approach to alert detection in system logs should be considered in future research.

The problem of unstructured data still remains a hinderance. The National Institute of Standards and Technology (NIST) states that this a major problem confronting event log management and asserts that there is still no agreed standard for terminology used in describing log events [27]. In IPLoM, this thesis has developed an effective algorithm for extracting the message types which can be used to mitigate this problem. However, this problem need not exist if more work is put into the development of a standard formalism for what and how information should be reported in system logs. The *syslog* format remains the only widely accepted standard for reporting information in system logs [38]. While the *syslog* standard defines a well defined schema for reporting log events, it leaves much to be desired with regard to the problem of unstructured data in system logs. Other proprietary attempts to develop a log reporting standard include: the Common Event Format (CEF) [24] and

Common Event Expression (CEE) [10]. Until such a standard is developed, the problem of lack of structure in the message part of event logs will persist and will continue to be a hindrance to automatic log analysis. Future research efforts in alert detection will be enhanced greatly if an *open source* log reporting standard which deals with the problem of free form event descriptions can be developed and adopted. However, the research performed in this thesis is still valid because even if such a standard is developed, message type extraction will be necessary to support legacy systems.

Automatic alert detection is meant to enable the goal of self-healing in autonomic computing by aiding the automatic diagnosis of fault conditions. However, not all faults leave traces in system logs. This implies that alert detection in system logs has limited scope for fault diagnosis and that complete fault diagnosis can be achieved using logs alone. This thesis has focussed on alert detection in system logs because previous efforts have shown them to be important for alert detection, but it does not argue that the proposed framework is the only or the best means for detecting alerts on systems. Other means which have been used effectively for this purpose include system metrics and system activity paths [8, 6], which are only part of the picture and cannot be relied on solely. Alert detection research should not focus on just one of these methods, as they are all important. Since there is a variety of alerts in system logs, an ensemble learning approach which combines the output of alert signatures learnt from each of these data sources would provide the most robust means of detecting all of the alerts on a system effectively.

The proposed alert detection framework can benefit from further improvements, which will enhance its accuracy and usability. Some of these improvements are suggested below and should prove interesting for future research in this direction.

One of the objectives of this thesis was to develop a framework which utilized minimum apriori information. This has been accomplished in that the only significant apriori information required by the system is the similarity categories for the event sources on which the framework must base its analysis. The process of identifying similar event sources can be automated. Similar event sources, discovered automatically, may lead to even more accuracy. It is suggested that similarity be based on the message types produced by a source and the information content of the message type with regard to the source. These features can be used to cluster event sources so that sources which produce the same message types in the same manner are grouped together. These clusters will form the similar categories for alert detection analysis.

The framework requires the setting of several parameters for its various components. These parameters are not a hindrance on the contrary, they enhance the flexibility of the framework. However, the ability to set these parameters automatically will enhance the usability of the framework, allowing the administrator to set only those goals which need to be achieved. This will be an interesting direction for future research. Such research will fall under self-configuration and self-optimization in autonomic computing. Indeed this supports the assertion that the four self-* objectives of autonomic computing are not mutually exclusive.

The STAD framework uses a rule base to detect anomalous log partition clusters. For now this rule base has been developed manually. The automatic discovery of rules to detect anomalous clusters is a possibility. Exploring this will be an interesting future research direction. A preliminary attempt using C5.0 rule induction is provided in Chapter 6. The results indicate that this is not a trivial problem, but C5.0's success on some of the datasets demonstrates that automatic rule discovery is possible. Suggestions on how automatic rule discovery research can continue from the point left off by this thesis is provided in Chapter 6.

A secondary objective of the framework is self-protection. The framework as implemented in this thesis is geared toward the detection of anomalies which manifest themselves in the message types reported by components of the system. This suffices for detecting faults but may not be adequate for detecting anomalies, which relate to security incidents. This thesis is of the opinion that anomalies which relate to security incidents are more likely to occur in the message type variables instead. However, with a little modification, the framework should be able to detect such anomalies. This can be done by creating sub-clusters of the clusters created by the framework, using the variables reported by the message types in the spatio-temporal units of the cluster. After this is done, an identification step can be applied to the sub-clusters to reveal the anomalies, which relate to the reported message type variables.

Generally, the visualization component of the framework and the utilization of feedback by the anomaly detection component need to be explored further. LogView is suggested for the visualization component. However to enable it work with the alert detection framework, its hierarchy needs to be adjusted to include spatio-temporal partitions and clusters of spatio-temporal partitions. A user study involving a prototype of the framework would be necessary. Such a study would provide valuable design choices which would improve the visualization component of the framework. In addition, the user study could help to understand the various kinds of feedback which the system can receive and use to improve itself over time. Some suggestions in this direction are provided in Chapter 8.

Bibliography

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[2] M. Aharon, G. Barash, I. Cohen, and E. Mordechai. One Graph Is Worth a Thousand Logs: Uncovering Hidden Structures in Massive System Event Logs. *Lecture Notes in Computer Science*, 5781/2009:227–243, 2009.

[3] S. Amershi, A. Lee, B.and Kapoor, R. Mahajan, and C. Blaine. Human-Guided Machine Learning for Fast and Accurate Network Alarm Triage. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011, Barcleona, Spain)*, pages 2564–2569, July 2011.

[4] B.B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies. In *ACM Transactions on Graphics*, volume 21, pages 883–854, October 2002.

[5] C Borgelt. Efficient Implementations of Apriori and Eclat. In *Workshop Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL, USA)*, Aachen, Germany, 2003. CEUR Workshop Proceedings 90.

[6] M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based Failure and Evolution Management. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 23–23, Berkeley, CA, USA, 2004. USENIX Association.

[7] E. H Chi. A Taxonomy of Visualization Techniques using the Data State Reference Model. In *Infovis 2000*, pages 69 –75, 2000.

[8] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, SOSP '05, pages 105–118, New York, NY, USA, 2005. ACM.

[9] J.E. Cook, Z. Du, C. Liu, and A.L. Wolf. Discovering models of behavior for concurrent workflows. *Comput. Ind.*, 53:297–319, April 2004.

[10] The MITRE Corporation. About Common Event Expression: CEE, A Standard Log Language for Event Interoperability in Electronic Systems. Published online at http://cee.mitre.org/about.html., 2011 October.

[11] W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J. Morar. Web Services Navigator: Visualizing the Execution of Web Services. *IBM Systems Journal*, 44(4):821–845, 2005.

[12] S. Deerwester, S.T. Dumais, G. Furnas, T.K. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[13] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey. Fulfilling the Vision of Autonomic Computing. *Computer, Monthly publication of the IEEE Computer Society*, 43(1):35–41, January 2010.

[14] J. Fekete. The Infovis Toolkit. In *Infovis 2004*, pages 167 – 174, 2004.

[15] Q. Fu, J Lou, Wang Y., and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on*, pages 149 –158, December 2009.

[16] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self adaptation with reusable infrastructure. *IEEE Computer*, 37(10):46–54, October 2004.

[17] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[18] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.

[19] J. Heer, S.K. Card, and J.A. Landay. prefuse: A Toolkit for Interactive Information Visualization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing*, pages 42–430, New York, 2005.

[20] G.R. Hendry and S.J. Yang. Intrusion Signature Creation via Clustering Anomalies. In *SPIE Conference on Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2008*, volume 6973, pages 69730C–69730C12, 2008.

[21] L. Huang, X. Ke, K. Wong, and S. Mankovskii. Symptom-based problem determination using log data abstraction. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '10, pages 313–326, New York, NY, USA, 2010. ACM.

[22] M.C. Huebscher and J.A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3):1–28, 2008.

[23] IBM. TEIRESIAS: Sequence Pattern Discovery. Published to web at http://cbcsrv.watson.ibm.com/Tspd.html. Last checked, April 2012.

[24] ArcSight Inc. Common event format - arcsight. Published online at http://www.arcsight.com/solutions/solutions-cef/. Last checked, April 2012.

[25] M. Jiang, M.A. Munawar, T. Reidemeister, and P.A.S Ward. Dependency-aware fault diagnosis with metric-correlation models in enterprise software systems. In *Proceedings of the 6th International Conference on Network and Service Management (CNSM)*, pages 137 – 141, October 2010.

[26] K. Julisch. Clustering Intrusion Detection Alarms to Support Root Cause Analysis. *ACM Transactions on Information and System Security*, 6(4):443–471, November 2003.

[27] K. Kent and M. Souppaya. *Guide to Computer Security Management*. National Institute of Standards and Technology, Gaithersburg, MD 20899-8930. USA, 800-92 edition, September 2006.

[28] J.O. Kephart and D.M. Chess. The Vision of Autonomic Computing. *Computer, Monthly publication of the IEEE Computer Society*, 36:41– 50, June 2003.

[29] A. Khalid, M.A. Haye, M.J. Khan, and S. Shamail. Survey of frameworks, architectures and techniques in autonomic computing. In *Autonomic and Autonomous Systems, 5th International Conference on*, volume 0, pages 220–225, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[30] M. Klemettinen. *A Knowledge Discovery Methodology for Telecommunications Network Alarm Databases*. PhD thesis, University of Helsinki, 1999.

[31] M.I. Krzywinski, J.E. Schein, I Birol, J. Connors, R. Gascoyne, D. Horsman, S.J. Jones, and M.A. Marra. Circos: An information aesthetic for comparative genomics. *Genome Research*, 2009.

[32] D. Levy and R. Chillarage. Early Warning of Failures through Alarm Analysis - A Case Study In Telecom Voice Mail Systems. In *IEEE International Symposium on Software Reliability Engineering*, pages 1–10, 2003.

[33] T. Li, F. Liang, S. Ma, and W. Peng. An Integrated Framework on Mining Log Files for Computing System Management. In *Proceedings of ACM KDD 2005*, pages 776–781, 2005.

[34] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, and M. Gupta. Filtering Failure Logs for a BlueGene/L Prototype. In *International Conference on Dependable Systems and Networks*, pages 476–485, July 2005 2005.

[35] Y. Liao and V.R. Vemuri. Using Text Categorization Techniques for Intrusion Detection. In *11th USENIX Security Symposium*, pages 51–59, August 2002.

[36] C. Lim, N. Singh, and S. Yajnik. A Log Mining Approach to Failure Analysis of Enterprise Telephony Systems. In *Proceedings of The 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2008)*, June 2008.

[37] Michael E. Locasto. Self-healing: science, engineering, and fiction. In *NSPW '07: Proceedings of the 2007 Workshop on New Security Paradigms*, pages 43–48, New York, NY, USA, 2008. ACM.

[38] C. Lonvick. The BSD Syslog Protocol. RFC3164, August 2001.

[39] LLC Los Alamos National Security. Operational Data to Support and Enable Computer Science Research. Published online at http://www.pdl.cmu.edu/FailureData/ and http://institutes.lanl.gov/data/fdata/. Accessed, April 2012.

[40] S. Ma and J.L. Hellerstein. Mining Partially Periodic Event Patterns with Unknown Periods. In *Proceedings of the 16th International Conference on Data Engineering*, pages 205–214, 2000.

[41] A. Makanju, S. Brooks, A.N. Zincir-Heywood, and E.E. Milios. Logview: Visualizing Event Log Clusters. In *Proceedings of Sixth Annual Conference on Privacy, Security and Trust (PST)*, pages 99 – 108, October 2008.

[42] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios. Fast Entropy Based Alert Detection in Super Computer Logs. In *International Conference on Dependable Systems and Networks Workshops (DSN-W)*, DSNW '10, pages 52–58, Washington, DC, USA, June 2010. IEEE Computer Society.

[43] A. Makanju, A. N. Zincir-Heywood, and E.E. Milios. A Lightweight Algorithm for Message Type Extraction in System Application Logs. *IEEE Transactions on Knowledge and Data Engineering*, June 2011.

[44] A. Makanju, A.N. Zincir-Heywood, and Milios. E.E. Message type extraction based alert detection in system logs. Technical Report CS-2009-08, Dalhousie University, March 2009.

[45] A. Makanju, A.N. Zincir-Heywood, and E.E. Milios. Clustering Event Logs Using Iterative Partitioning. In *Proceedings of the 15th ACM Conference on Knowledge Discovery in Data*, pages 1255–1264, July 2009.

[46] A. Makanju, A.N. Zincir-Heywood, and E.E. Milios. Extracting Message Types from BlueGene/L's Logs. In *Proceedings of the SOSP Workshop on the Analysis of System Logs (WASL)*, 2009.

[47] A. Makanju, A.N. Zincir-Heywood, and E.E. Milios. An Evaluation of Entropy Based Approaches to Alert Detection in High Performance Cluster Logs. In *Proceedings of the 7th International Conference on Quantitative Evaluation of SysTems (QEST)*, pages 69–78, September 2010.

[48] A. Makanju, A.N. Zincir-Heywood, and E.E. Milios. A Next Generation Entropy-based framework for Alert Detection in System Logs. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 626 –629, May 2011.

[49] A. Makanju, A.N. Zincir-Heywood, and E.E. Milios. Storage and Retrieval of System Log Events using a Structured Schema based on Message Type Transformation. In *Proceedings of the 26th ACM Symposium on Applied Computing (SAC)*, pages 528 – 533, March 2011.

[50] A. Makanju, A.N. Zincir-Heywood, and E.E. Milios. System State Discovery via Information Content Clustering of System Logs. In *Proceedings of the 2011 International Conference on Availability, Reliability and Security, ARES 2011. Vienna, Austria.*, pages 301–306, August 2011.

[51] A. Makanju, A.N. Zincir-Heywood, and E.E. Milios. Spatio-Temporal Decomposition, Clustering and Identification for Alert Detection in System Logs. In *Proceedings of the 27th ACM Symposium on Applied Computing*, SAC '12, New York, NY, USA, March 2012. ACM.

[52] A. Makanju, A.N. Zincir-Heywood, and E.E. Milios. Spatio-Temporal Decomposition, Clustering and Identification for Alert Detection in System Logs. In *Proceedings of the 13th IEEE/IFIP Network Operations and Management Symposium*, NOMS '12. IEEE Computer Society, April 2012.

[53] J. Miao, M.A. Munawar, T. Reidemeister, and P.A.S. Ward. Automatic Fault Detection and Diagnosis in Complex Software Systems by Information-Theoretic Monitoring. In *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, pages 285 –294, July 2009.

[54] U. o. M. HCI Lab. Piccolo Toolkit: A Structured 2D Graphics Framework. Published online at http://www.piccolo2d.org/. Accessed, April 2012.

[55] A. Oliner, A. Aiken, and J. Stearley. Alert Detection in System Logs. In *Proceedings of the International Conference on Data Mining (ICDM). Pisa, Italy.*, pages 959–964, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[56] A. Oliner and J. Stearley. What Supercomputers say: A Study of Five System Logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007 (DSN '07)*, pages 575–584, June 2007.

[57] W. Peng, C. Perng, T. Li, and H. Wang. Event summarization for system management. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 1028–1032, New York, NY, USA, 2007. ACM.

[58] C. Perng, D. Thoenen, G. Grabarnik, S. Ma, and J. Hellerstein. Data-driven validation, completion and construction of event relationship networks. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 729–734, New York, NY, USA, 2003. ACM.

[59] Wikimedia Project. C4.5 Algorithm. Published online at http://en.wikipedia.org/wiki/C4.5_algorithm. Accessed, April 2012.

[60] J.R. Reuning. Applying Term Weight Techniques to Event Log Analysis for Intrusion Detection. Master's thesis, University of North Carolina at Chapel Hill, July 2004.

[61] I. Rigoutsos and A. Floratos. Combinatorial Pattern Discovery in Biologcal Sequences: The Teiresias Algorithm. In *BioInformatics*, volume 14, pages 55–67. Oxford University Press, 1998.

[62] Ross Quinlan. C5.0: An Informal Tutorial. Published online at http://www.rulequest.com/see5-unix.html. Accessed, April 2012.

[63] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset. Analyzing System Logs: A New View of What's Important. In *Proceedings of the 2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, pages 6:1–6:7, Berkeley, CA, USA, 2007. USENIX Association.

[64] F. Salfener and M. Malek. Using Hidden Semi-Markov Models for Effective Online Failure Prediction. In *26th IEEE International Symposium on Reliable Distributed Systems.*, pages 161–174, 2007.

[65] B. Shneiderman. Tree Visualization with Tree-Maps: A 2-D space filling approach. In *ACM Transactions on Graphics.*, volume 2, pages 92–99, 1992.

[66] Sourceforge.net. The prefuse Visualization Toolkit. Published online at http://www.prefuse.org. Last Accessed,, April 2012.

[67] Splunk Inc. Splunk: Operational Intelligence Software. Published online at http://www.splunk.com/product. Accessed, April 2012.

[68] J. Stearley. Towards Informatic Analysis of Syslogs. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 309–318, 2004.

[69] J. Stearley and A.J. Oliner. Bad Words: Finding Faults in Spirit's Syslogs. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 765 –770, May 2008.

[70] Jon Stearley. Sisyphus Log Data Mining Toolkit. Published online at http://www.cs.sandia.gov/sisyphus. Accessed, April 2012.

[71] R. Sterritt. Towards autonomic computing: effective event management. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pages 40–47, Dec. 2002.

[72] D. Thoenen, J. Riosa, and J.L. Hellerstein. Event relationship networks: a framework for action oriented analysis in event management. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 593 –606, 2001.

[73] TOP500.Org. Top500 supercomputing sites. Published online at http://www.top500.org/. Last Accessed, April 2012.

[74] Brad Topol. Automating Problem Determination: A First Step Toward Self Healing Computing Systems. IBM White Paper, October 2003.

[75] USENIX. USENIX - The Computer Failure Data Repository. Published to web at http://cfdr.usenix.org/data.html. Last Accessed, April 2012.

[76] R. Vaarandi. A Data Clustering Algorithm for Mining Patterns from Event Logs. In *Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM)*, pages 119–126, 2003.

[77] R. Vaarandi. A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs. In *Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems (LNCS)*, volume 3283, pages 293–308, 2004.

[78] R. Vaarandi. Mining Event Logs with SLCT and Loghound. In *Proceedings of the 2008 IEEE/IFIP Network Operations and Management Symposium*, pages 1071–1074, April 2008.

[79] R. Vaarandi. LogHound - A Tool for Mining Frequent Patterns from Event Logs. Published online at http://ristov.users.sourceforge.net/loghound/. Accessed, April 2012.

[80] R Vaarandi. SLCT - Simple Logfile Clustering Tool. Published online at http://ristov.users.sourceforge.net/slct/. Accessed, April 2012.

[81] W. van der Aalst and H. Verbeek. Process Mining in Web Services: The Websphere Case. *IEEE Bulletin of the Technical Committee on Data Engineering*, 33(3):46–49, 2008.

[82] W. Xu. *Detecting Large Scale System Problems by Mining Console Logs.* PhD thesis, University of California, Berkeley, 2010.

[83] W. Xu, L. Huang, A. Fox, D. Patterson, and M.I. Jordan. Detecting large-scale system problems by mining console logs. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems principles*, pages 117–132, New York, NY, USA, 2009. ACM.

[84] Z. Xue, X. Dong, S. Ma, and W. Dong. A Survey on Failure Prediction of Large Scale Server Clusters. In *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 733–738, July 2007.

[85] M.J. Zaki. Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[86] Q. Zheng, K. Xu, W. Lv, and S. Ma. Intelligent Search for Correlated Alarm from Database Containing Noise Data. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 405–419, 2002.

Appendix A

A.1 Tool Documentation

A detailed listing and definitions of the options of the open source tools used in this thesis can be found in this Appendix.

A.1.1 Full List of SLCT Options

```
-b <byte offset>
    when processing the input file(s), ignore the first
    <byte offset> bytes of every line. This option can be
    used to filter out the possibly irrelevant information in
    the beginning of every line (e.g., timestamp and hostname).
    The default value for the option is zero, i.e., no bytes
    are ignored.
-c <clustertable size>
    the number of slots in the cluster candidate hash table
    (note that this option does not limit the actual size of the
    hash table, since multiple elements can be connected
    to a single slot). The default value for  the option is
    (100 * the number of frequent words).
-d <regexp>
    the regular expression describing the word delimiter. The
    default value for the option is '[ \t]+', i.e., words are
    separated from each other by one or more space or
    tabulation characters.
-f <regexp>
    when processing the input file(s), ignore all lines that do not
    match the regular expression. The regular expression can
    contain ()-subexpressions, and when -t <template> option
    has also been given, the values of those subexpressions

    can be retrieved in <template> through $-variables. When
    -f and -b options are used together, the -b option is applied
    first.
-g <slice size>
    when the -j option has been given, SLCT inspects the table
    of candidates and compares each candidate with others, in
    order to find its subclusters. This task has quadratic
    complexity, and if the candidate table is larger, substantial
    amount of time could be required to complete the task. In
    order to reduce the time complexity, SLCT will divide
    candidate table into slices, with each slice having <slice size>
    candidates, and all candidates in the same slice having the
    same number of constant words in their descriptions. A
    descriptive bit vector is then calculated for every slice that lists
    all constant words the candidates of a given slice have. If SLCT
    is inspecting the cluster candidate C for subclusters, and the
    descriptive vector of the slice S does not contain all the
    constant words of the candidate C, the candidates from the
    slice S will be skipped (i.e., they will not be compared with
    the candidate C). If the -j option has been given, the default
    value for the -g option is
    (the number of cluster candidates / 100 + 1).
-i <seed>
    the value that is used to initialize the rand(3) based random
    number generator which is used to generate seed values
    for string hashing functions inside SLCT. The default value
    for the option is 1.
-j
```

```
    when processing the table of cluster candidates, also consider
    the relations between candidates, and allow the candidates
    (and clusters) to intersect. This option and the option -z are
    mutually exclusive, since -j requires the presence of all
    candidates in order to produce correct results, but with -z not all
    candidates are inserted into the candidate table.
-o <outliers file>
    the file where outliers are written. This option is meaningless without
    the -r option.
-r
    after the clusters have been found from the set of candidates, refine
    the cluster descriptions.
-t <template>
    template that is used to convert input lines, after they have matched the
    regular expression given with the -f option. Template is a string that will
    replace the original input line, after the $-variables in the template have
    been replaced with the values of ()-subexpressions from the regular
    expression. For example, if the regular expression given with the -f
    option is 'sshd\[[0-9]+\]: (.+)', and the template is "$1", then the line
    sshd[1344]: connect from 192.168.1.1
    will be converted to
    connect from 192.168.1.1
    This option is meaningless without the -f option.
-v <wordvector size>
    the size of the word summary vector. The default value for the option is
    zero, i.e., no summary vector will be generated.
-w <wordtable size>
    the number of slots in the vocabulary hash table. The default value for
    the option is 100,000.
-z <clustervector size>
    the size of the cluster candidate summary vector. The default value for
    the option is zero, i.e., no summary vector will be generated. This
```

option and the option -j are mutually exclusive, since -j requires the presence of all candidates in order to produce correct results, but when the summary vector is employed, not all candidates are inserted into the candidate table.

-s <support>

the support threshold value. The value can be either an integer, or a real number with a trailing %-sign.

A.1.2 Full List of Loghound Options

-b <byte offset>

when processing the input file(s), ignore the first <byte offset> bytes of every line. This option can be used to filter out the possibly irrelevant information in the beginning of every line (e.g., timestamp and hostname). The default value for the option is zero, i.e., no bytes are ignored.

-c

output closed frequent itemsets only.

-d <regexp>

the regular expression describing the item delimiter in input file(s). The default value for the option is '[\t]+', i.e., items are separated from each other by one or more space or tabulation characters.

-f <regexp>

when processing the input file(s), ignore all lines that do not match the regular expression. The regular expression can contain ()-subexpressions, and when -t <template> option has also been given, the values of those subexpressions can be retrieved in <template> through $-variables. When -f and -b options are used together, the -b option is applied first.

-g

assume that each line in input file(s) represents a set of events, and mine frequent event type patterns from the file(s). If this option is omitted, it is assumed that input file(s) are raw event log(s), and frequent line patterns will be mined from the file(s).

```
-i <seed>
    the value that is used to initialize the rand(3) based random
    number generator which is used to generate seed values for
    hashing functions inside LogHound. The default value for the
    option is 1.
-l <max itemset size>
    don't mine itemsets containing more than <max itemset size>
    items.
-n <cache trie support>
    create a cache trie that is guaranteed to contain transaction
    itemsets present at least <cache trie support> times in the data set.
    The default value for the option is zero, i.e., load the entire input
    data set into main memory.
-o <out-of-cache file>
    The location of the out-of-cache file. When this option is omitted,
    the entire input data set is loaded into main memory.
-t <template>
    template that is used to convert input lines, after they have
    matched the regular expression given with the -f option.
    Template is the string that will replace the original input line,
    after the $-variables in the template have been replaced with
    the values of ()-subexpressions from the regular expression.
    For example, if the regular expression given with the -f option
    is 'sshd\[[0-9]+\]: (.+)', and the template is "$1", then the line
    sshd[1344]: connect from 192.168.1.1
    will be converted to
    connect from 192.168.1.1
    before the line will be processed by the mining algorithm that is
    built into LogHound. This option is meaningless without the -f option.
-v <item vector size>
    the size of the item summary vector. The default value for the
    option is zero, i.e., no summary vector will be generated.
```

```
-w <item table size>
    the number of slots in the item hash table. The default value for
    the option is 100,000.
-z <transaction vector size>
    the size of the transaction summary vector. If this option is
    omitted or zero is specified for its value, the summary vector
    of size (the number of frequent items * 100) is used. This
    option is meaningless without the -n or the -o option, or
    when zero is specified for the value of the -n option.
-s <support>
    the support threshold value. The value can be either
    an integer, or a real number with a trailing %-sign.
```

A.1.3 Full List of C5.0 Options

```
-f <filestem>
   select the application
-s
   partition discrete values into subsets
-r
    generate rule-based classifiers
-u <bands>
    sort rules by their utility into bands
-b
    use boosting with 10 trials
-t  <trials>
   use boosting with the specified number of trials
-w
    winnow the attributes before constructing a classifier
-p
    use soft thresholds
-g
    do not use global tree pruning
```

```
-c <CF>
    set the CF value for pruning trees
-m <cases>
    set the minimum cases for at least two branches of a split
-S <x>
   use a sample of x\% for training and a disjoint sample for testing
-I <seed>
   set the sampling seed value
-X <folds>
    carry out a cross-validation
-e
    ignore any costs file
-h
    print a short summary of the options
```

A.1.4 Full List of Apriori Options

```
-t#      target type                            (default: s)
         (s: frequent item sets, c: closed item sets,
          m: maximal item sets,  r: association rules)
-m#      minimum number of items per set/rule     (default: 1)
-n#      maximum number of items per set/rule     (default: no limit)
-s#      minimum support    of a     set/rule     (default: 10%)
-S#      maximum support    of a     set/rule     (default: 100%)
         (positive: percentage, negative: absolute number)
-c#      minimum confidence of a     rule         (default: 80%)
-o       use the original rule support definition (body & head)
-e#      additional evaluation measure            (default: none)
-a#      aggregation mode for evaluation measure  (default: none)
-z       zero evaluation below expected support   (default: evaluate all)
-d#      minimum value of add. evaluation measure (default: 10%)
-I#      minimum increase of evaluation measure   (default: no limit)
         (not applicable with evaluation averaging, i.e. option -aa)
```

```
-p#      (min. size for) pruning with evaluation  (default: no pruning)
         (< 0: backward,   > 0: forward)
-l#      sort item sets in output by their size   (default: no sorting)
         (< 0: descending, > 0: ascending)
-g       write item names in scannable form (quote certain characters)
-h#      record header  for output                (default: "")
-k#      item separator for output                (default: " ")
-i#      implication sign for association rules   (default: " <- ")
-v#      output format for set/rule information   (default: "  (%1S)")
-q#      sort items w.r.t. their frequency        (default: 2)
         (1: ascending, -1: descending, 0: do not sort,
          2: ascending, -2: descending w.r.t. transaction size sum)
-u#      filter unused items from transactions    (default: 0.1)
         (0: do not filter items w.r.t. usage in sets,
         <0: fraction of removed items for filtering,
         >0: take execution times ratio into account)
-j       use quicksort to sort the transactions   (default: heapsort)
-x       do not prune with perfect extensions     (default: prune)
-y       a-posteriori pruning of infrequent item sets
-T       do not organize transactions as a prefix tree
-w       integer transaction weight in last field (default: only items)
-r#      record/transaction separators            (default: "\n")
-f#      field /item       separators             (default: " \t,")
-b#      blank   characters                       (default: " \t\r")
-C#      comment characters                       (default: "#")
-!       print additional option information
infile   file to read transactions from           [required]
outfile  file to write item sets/assoc. rules to  [optional]
appfile  file stating a selection of items        [optional]
         or item appearance indicators (for association rules)
```

Appendix B

B.1 Comparing Entropy-Based Approaches in Online Alert Detection

This appendix presents the results of evaluations which are intended to demonstrate how well the proposed entropy-based alert detection mechanisms would work if applied online for a production system. Alert detection is essentially an online task, so these evaluations are necessary and differ from the evaluations done in Chapter 4 through the use of sliding windows to simulate online activity. For a window of size W, the nodeinfo score for each nodehour is computed using only data seen in the $W - 1$ days prior to the day on which the nodehour occurred, each day starting at 12:00 AM.

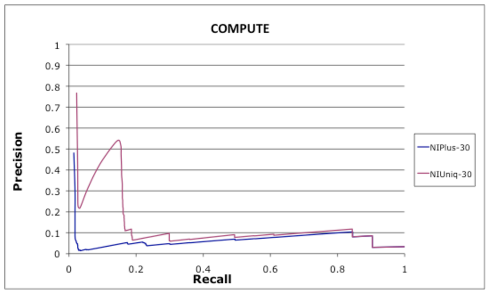
The evaluations were performed on the datasets listed in Table 4.7 in Chapter 4. The *NIPlus* method introduced in Chapter 4 was compared as a baseline against the *NIUniq* method introduced in Chapter 5. Previous off line evaluations have shown that *NIPlus* will perform as well as Nodeinfo with one hundred times less computation [42], so it suffices as a baseline. Another set of evaluations indicates that *NIUniq* performs better for offline alert detection than *NIPlus* as long as the alerts do not have a *bursty* signature [48]. This evaluation complements previous evaluations by showing how the methods work online. A third method, called $NIMax$, is not included here due to its lack of robustness [48].

Window sizes of 30, 60 and 90 days were considered in the evaluations, while the first 90 days of each dataset were omitted from the evaluation but were used in building the model. Two sets of evaluations were performed. In the first, results of running *NIPlus* and *NIUniq* against all the datasets using a window size of 30 were compared. This evaluation would help to determine how *NIUniq* would perform against the baseline in production environment. The second evaluation compares the results of running *NIUniq* using window sizes of $30, 60$ and 90 days. This evaluation would help to determine what effect the size of the window would have on detection accuracy.

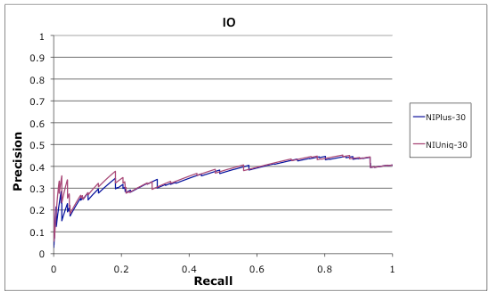
The evaluation metrics are Recall, Precision and F-Measure as defined in Eqs. 3.4, 3.3 and 3.5 respectively, in Chapter 3 and FPR as defined in Eq. 6.4 defined in Chapter 6.

B.1.1 Results

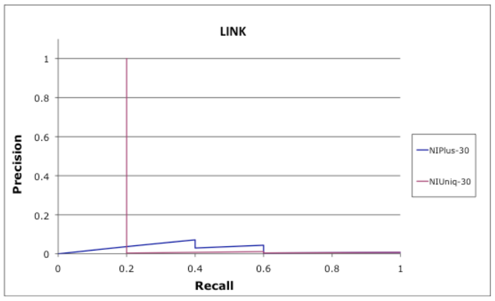
The Precision-Recall plots for the first evaluation are shown in Figs. B.5, B.6, B.7 and B.8. These plots show that, apart from the Tbird-Compute dataset, *NIPlus* does not show any dominance over *NIUniq* for Recall rates above 30%. *NIUniq* achieves marginal dominance over *NIPlus* in three datasets, i.e. the BGL-Compute, BGL-Other, Liberty-Compute datasets. However, *NIUniq* dominates *NIPlus* significantly in five datasets, i.e. the Liberty-Admin, Liberty-Other, Spirit-Admin, Spirit-Other and Tbird-Other datasets. The approaches achieved similar performance in four datasets: the BGL-IO, BGL-Link, Spirit-Compute and Tbird-Admin datasets.



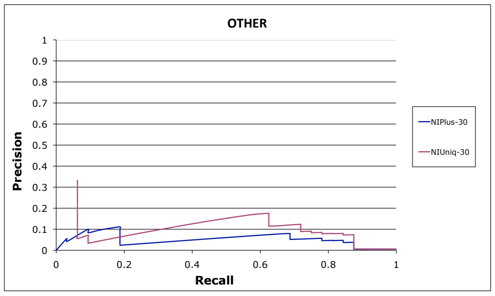
(a) BGL-Compute



(b) BGL-IO

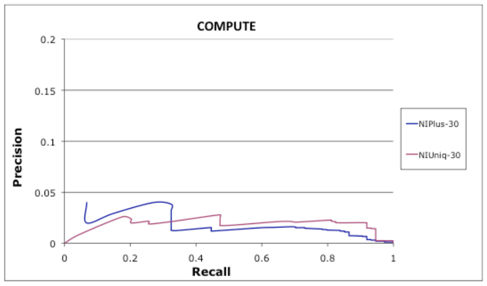


(c) BGL-Link

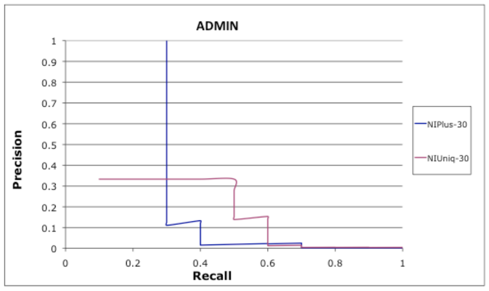


(d) BGL-Other

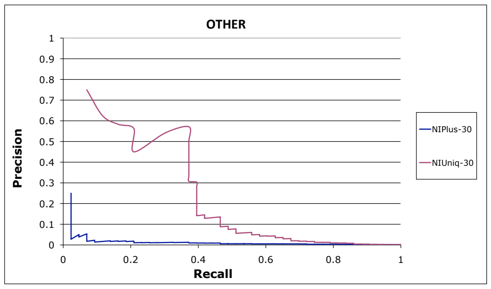
Figure B.1: Precision-Recall plots for the BGL node categories. Each plot compares the result of using the original nodeinfo framework with message types incorporated (NIPlus) and nodeinfo based on Eq. B.1 (NIUniq). A window size of 30 was used for both methods.



(a) Liberty-Compute



(b) Liberty-Admin

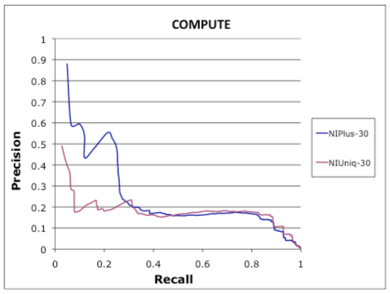


(c) Liberty-Other

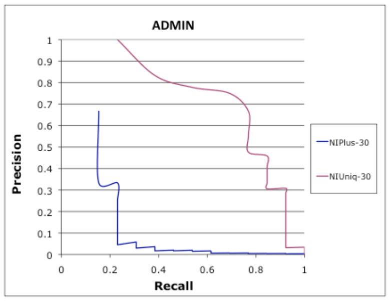
Figure B.2: Precision-Recall plots for the Liberty node categories. Each plot compares the result of using the original nodeinfo framework with message types incorporated (NIPlus) and nodeinfo based on Eq. B.1 (NIUniq). A window size of 30 was used for both methods.

The Precision-Recall plots for the second evaluation are shown in Figs. B.5, B.6, B.7 and B.8. As in the previous evaluation the level of accuracy differs among the different datasets. However, it can be noticed that the results within each dataset do not differ much from each other except in five datasets: the BGL-IO, Liberty-Compute, Liberty-Admin, Liberty-Other, Spirit-Admin datasets. In these datasets, the trend suggests that larger window sizes give better results. These results are not surprising as it should be expected that models built using more examples should be more robust than those built using fewer examples.

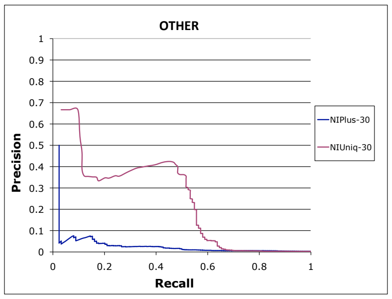
The bar charts in Figs. B.9, B.10, B.11 and B.12 show the FPRs achieved at 90% Recall and the maximum F-Measure for all of the evaluations. The results indicate that in certain cases very high FPRs were experienced at 90% Recall, meaning that the methods yield poor results for high detection rates. This is attributed to the presence of a few *outlier* nodehours [47]. *Outliers* are alert nodehours with very



(a) Spirit-Compute



(b) Spirit-Admin



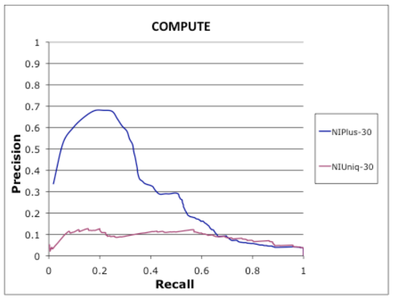
(c) Spirit-Other

Figure B.3: Precision-Recall plots for the Spirit node categories. Each plot compares the result of using the original nodeinfo framework with message types incorporated (NIPlus) and nodeinfo based on Eq. B.1 (NIUniq). A window size of 30 was used for both methods.

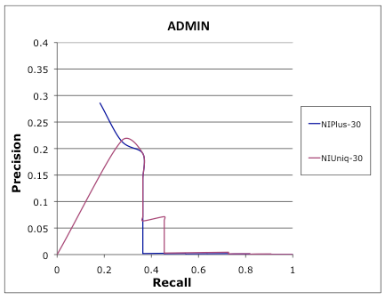
low nodeinfo scores which end up being placed very low in the ranking. A possible approach to mitigating this problem can be found in [48]. Augmenting the nodeinfo score using other metrics is suggested.

However, operationally acceptable FPRs are achieved at maximum F-Measure except in the case of the BGL-IO dataset. These results suggest that we can expect good results from the mechanisms as long as they are primed to detect alerts while maintaining a balance between high Recall and high Precision rates. The high FPRs achieved at maximum F-Measure for the BGL-IO category are due to the fact that most of the alerts in this category are not linked to the reporting node directly [47]. The alerts are due to a shared component whose faults are detected and reported by all the nodes, hence these alerts appear to be distributed evenly across all nodes.

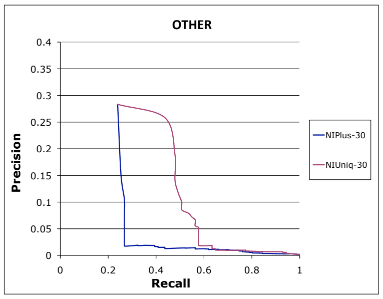
It is important to remember that low Precision scores (below 50%) in the Precision-Recall plots presented do not mean poor results necessarily [48]. The fact that alert



(a) Tbird-Compute



(b) Tbird-Admin



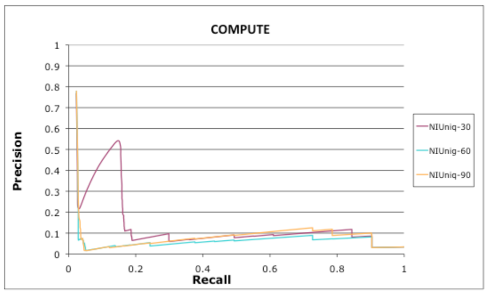
(c) Tbird-Other

Figure B.4: Precision-Recall plots for the Tbird node categories. Each plot compares the result of using the original nodeinfo framework with message types incorporated (NIPlus) and nodeinfo based on Eq. B.1 (NIUniq). A window size of 30 was used for both methods.

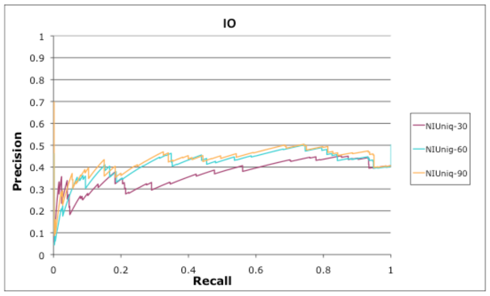
nodehours form less than 1% of all nodehours in some of the datasets make achieving high Precision scores difficult. At this level, Precision values are very sensitive to even small changes in the value of k. Even at low Precision scores the methods perform in most cases at least ten times better than what would be expected of a simple random selection of nodehours.

B.1.2 Conclusion

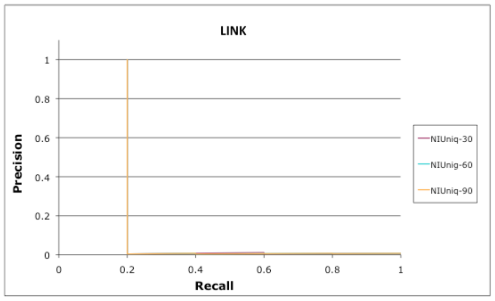
A summary comparison of *NIPlus* and *NIUniq* is given Table B.1. The data in the table shows the average score across all the datasets for the maximum precision and maximum F-Measure achieved for each experiment run. In addition, it shows the FPR achieved at the points when the maximum F-Measure and a Recall of 90% were achieved. The data in the table shows *NIUniq* doing better than *NIPlus* using all the metrics.



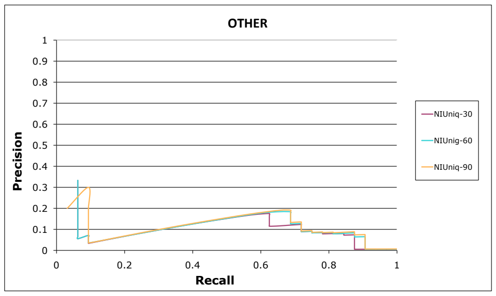
(a) BGL-Compute



(b) BGL-IO



(c) BGL-Link



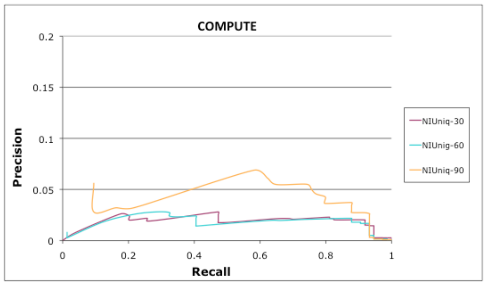
(d) BGL-Other

Figure B.5: Precision-Recall plots for the BGL node categories. Each plot compares the result of using the nodeinfo based on Eq. B.1 (NIUniq) while varying the window size to 30, 60 and 90 days.

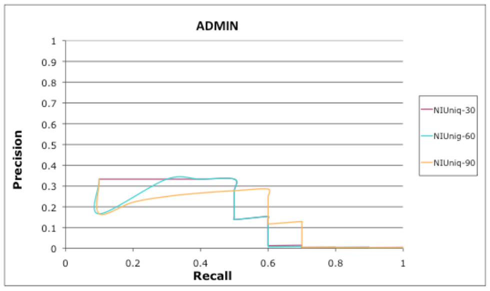
With this result, it can be concluded that *NIUniq* is a more robust detection method than *NIPlus*. Hence, its use as the method of choice in STAD is justified. Other factors make *NIUniq* appropriate for STAD but these are explained in Chapter 5.

A summary comparison of *NIUniq* using the different time windows is shown in Table B.2. The table shows the same metrics as those used in Table B.1. The table shows that the results achieved using a 90 day window outperforming all cases except for the FPR achieved at maximum F-Measure.

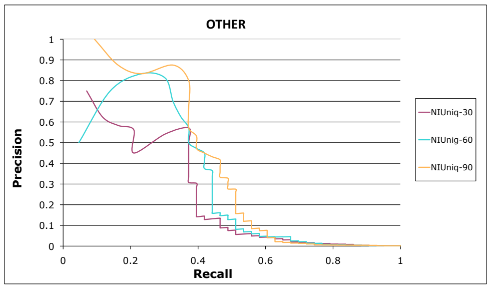
These results would suggest that marginal increases can be expected in a production environment from using larger size windows. Larger windows mean more data and hence more computation. However, the use of MTI in *NIUniq* provides considerable reduction in computation, implying that computation can be carried out fast enough even with larger size windows.



(a) Liberty-Compute



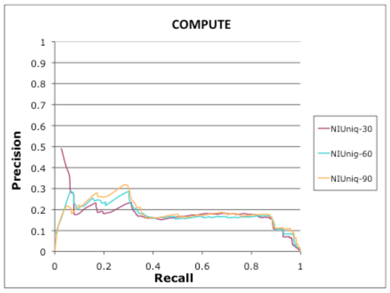
(b) Liberty-Admin



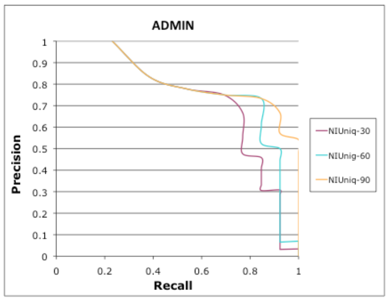
(c) Liberty-Other

Figure B.6: Precision-Recall plots for the Liberty node categories. Each plot compares the result of using the nodeinfo based on Eq. B.1 (NIUniq) while varying the window size to 30, 60 and 90 days.

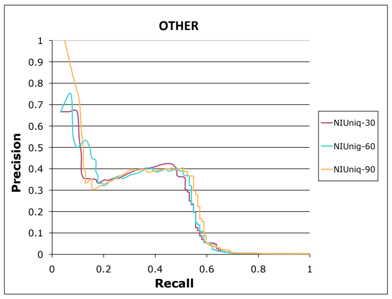
$$NodeInfo(H_j^c) = \sqrt{\sum_{w=1}^{|W|} (g_w * z_{w,j}^c)^2} \tag{B.1}$$



(a) Spirit-Compute



(b) Spirit-Admin



(c) Spirit-Other

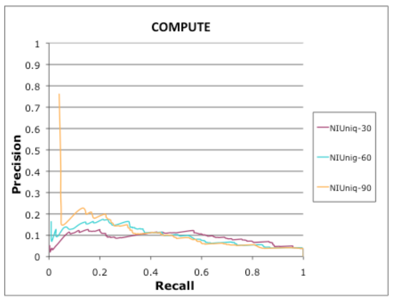
Figure B.7: Precision-Recall plots for the Spirit node categories. Each plot compares the result of using the nodeinfo based on Eq. B.1 (NIUniq) while varying the window size to 30, 60 and 90 days.

Table B.1: Average performance across all datasets for *NIPlus* and *NIUniq* using a window size of 30 days

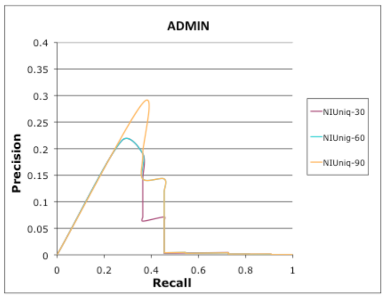
	NIPlus-30	**NIUniq-30**
Max. Precision	0.44	0.50
Max. F-Measure	0.25	0.35
FPR(Max. F-Measure)	0.09	0.06
FPR(90% Recall)	0.43	0.33

Table B.2: Average performance across all datasets for *NIUniq* using window sizes of 30, 60 and 90 days

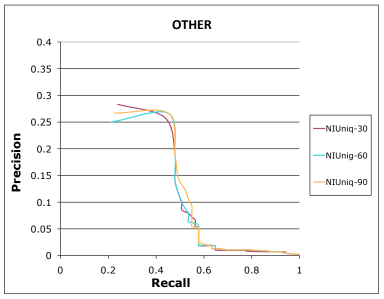
	NIUniq-30	**NIUniq-60**	**NIUniq-90**
Max. Precision	0.50	0.50	0.60
Max. F-Measure	0.35	0.35	0.38
FPR(Max. F-Measure)	0.06	0.06	0.07
FPR(90% Recall)	0.33	0.29	0.25



(a) Tbird-Compute

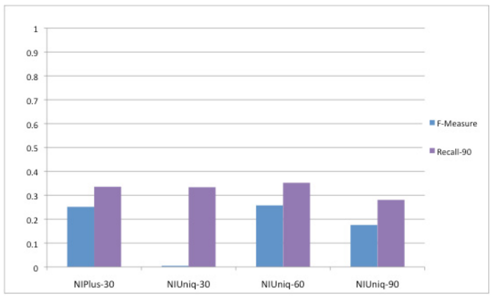


(b) Tbird-Admin

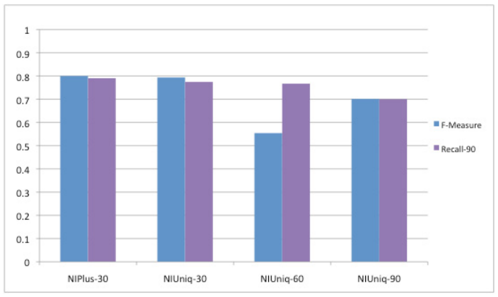


(c) Tbird-Other

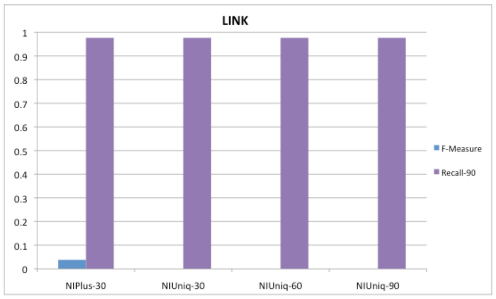
Figure B.8: Precision-Recall plots for the Tbird node categories. Each plot compares the result of using the nodeinfo based on Eq. B.1 (NIUniq) while varying the window size to 30, 60 and 90 days.



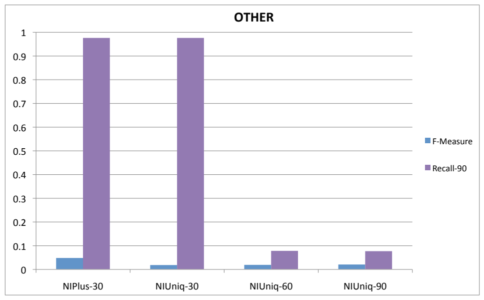
(a) BGL-Compute



(b) BGL-IO

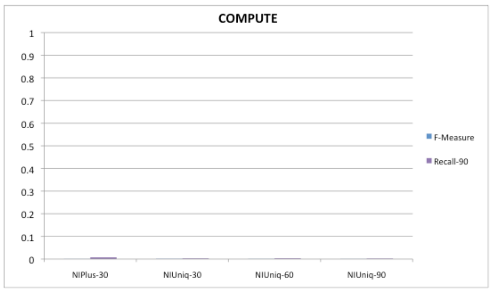


(c) BGL-Link

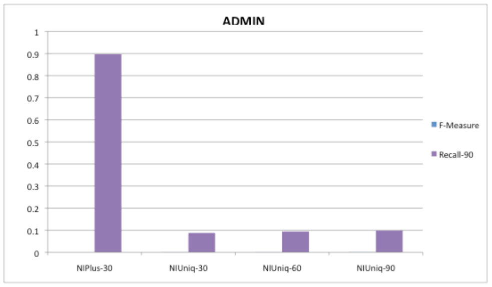


(d) BGL-Other

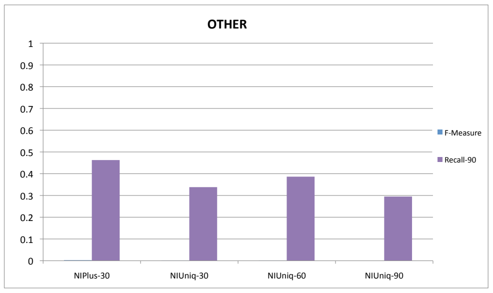
Figure B.9: The bar chart compares the FPR achieved at the maximum F-Measure score and at 90% Recall for the experiment trials on the BGL node categories shown in Figs. B.1, B.5.



(a) Liberty-Compute

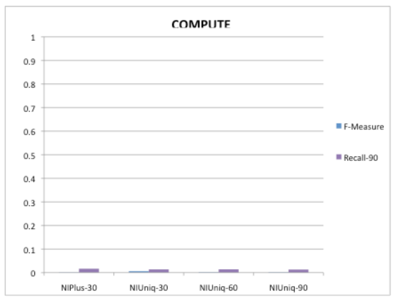


(b) Liberty-Admin

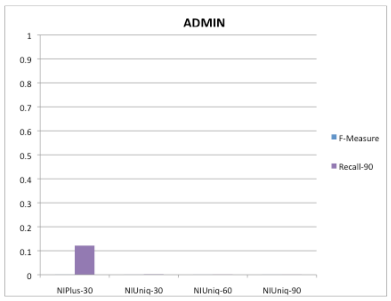


(c) Liberty-Other

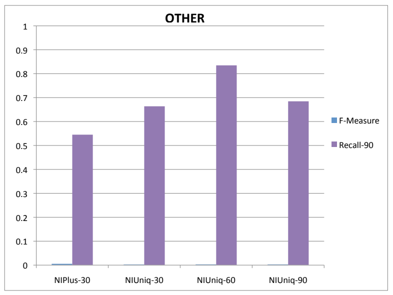
Figure B.10: The bar chart compares the FPR achieved at the maximum F-Measure score and at 90% Recall for the experiment trials on the Liberty node categories shown in Figs. B.2, B.6.



(a) Spirit-Compute

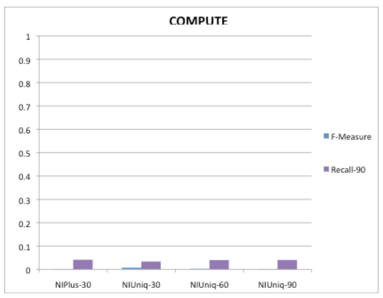


(b) Spirit-Admin

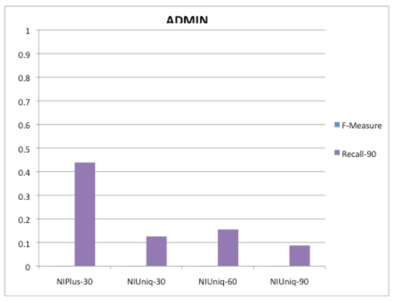


(c) Spirit-Other

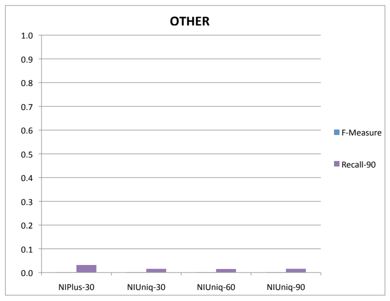
Figure B.11: The bar chart compares the FPR achieved at the maximum F-Measure score and at 90% Recall for the experiment trials on the Spirit node categories shown in Figs. B.3, B.7.



(a) Tbird-Compute



(b) Tbird-Admin



(c) Tbird-Other

Figure B.12: The bar chart compares the FPR achieved at the maximum F-Measure score and at 90% Recall for the experiment trials on the Tbird node categories shown in Figs. B.4, B.8.

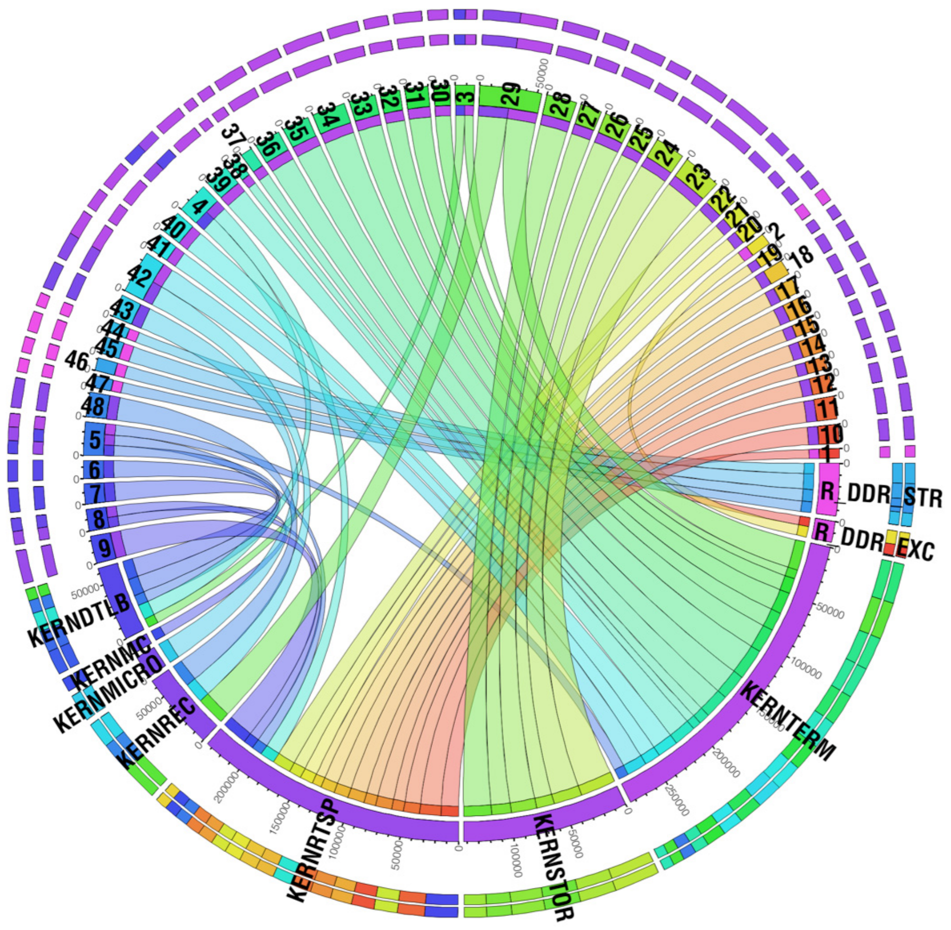
Appendix C

C.1 Further Results And Figures

This appendix contains results and/or figures of evaluations which are not included in the main text of the thesis.

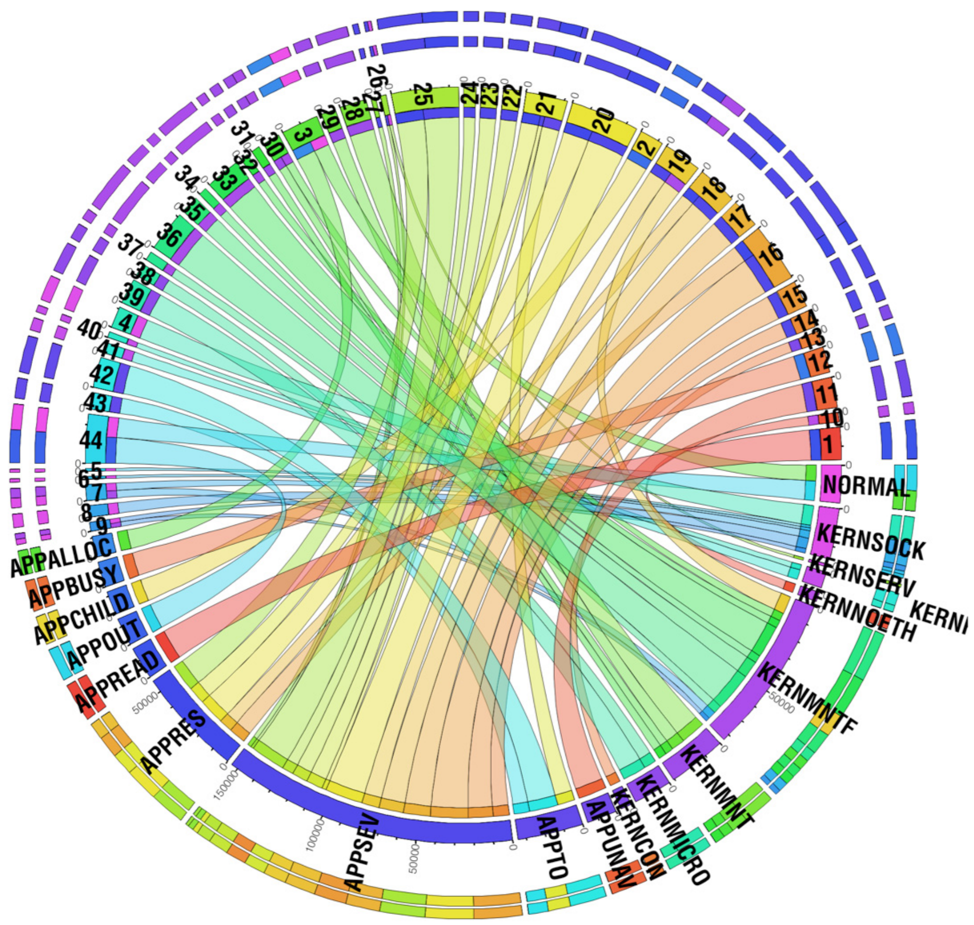
C.1.1 HPC Node Circos Plots

The circos plot visualizations for all the HPC node categories are shown in in Figs. C.1, C.5, C.6 and C.7. Clusters from only a single node are shown for the dissimilar node categories; the clusters from the other nodes are shown in proceeding section.



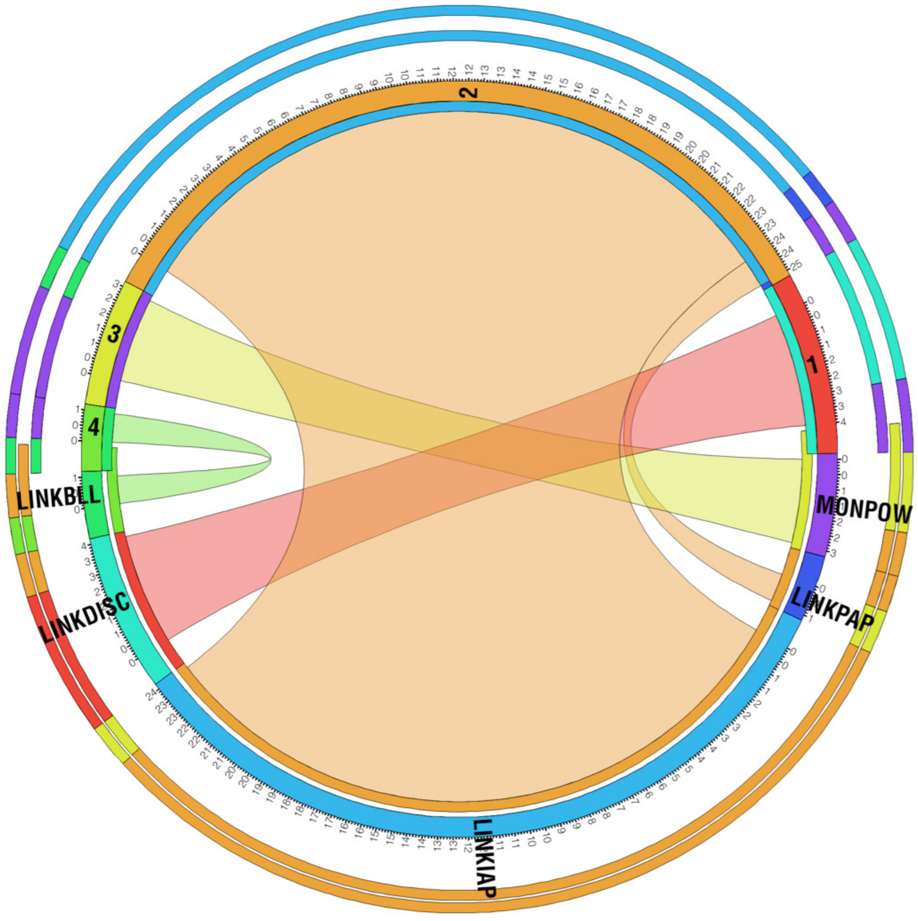
(a) BGL-Compute

Figure C.1: Circos Plots for nodes for the BGL node categories. Each plot shows the plot for clustering as performed on all nodehours in each category.



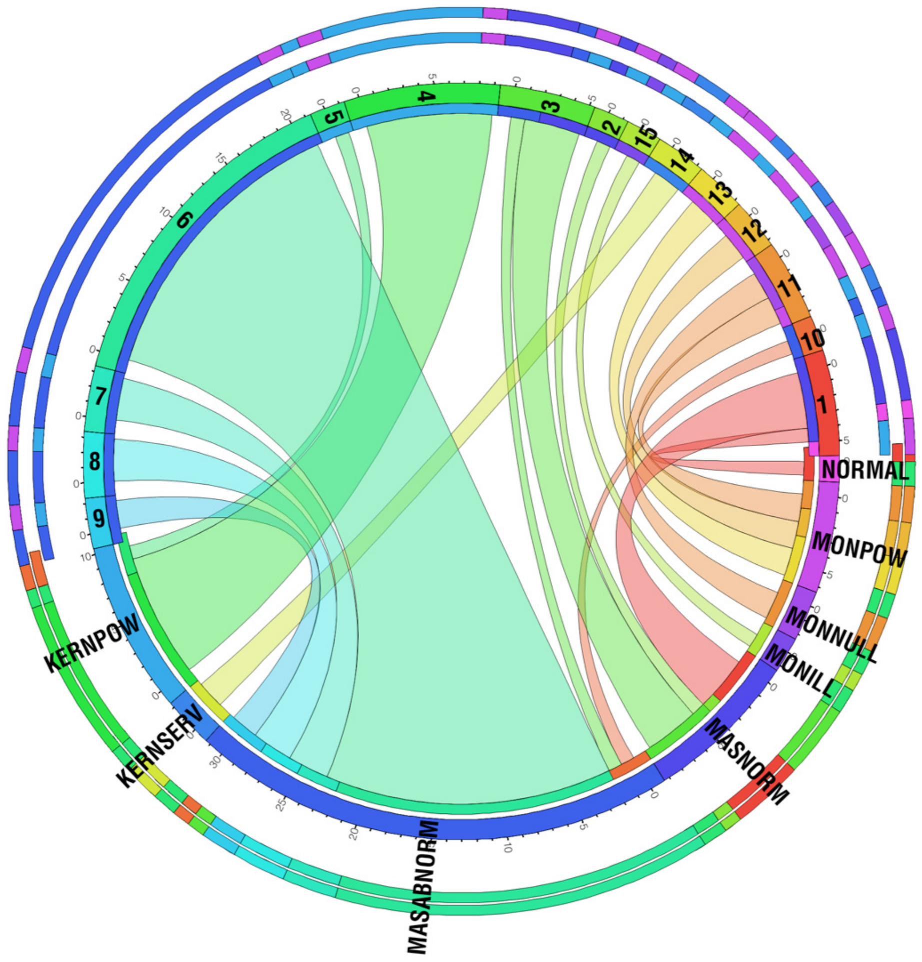
(b) BGL-IO

Figure C.1: Circos Plots for the BGL node categories. Each plot shows the plot for clustering as performed on all nodehours in each category.



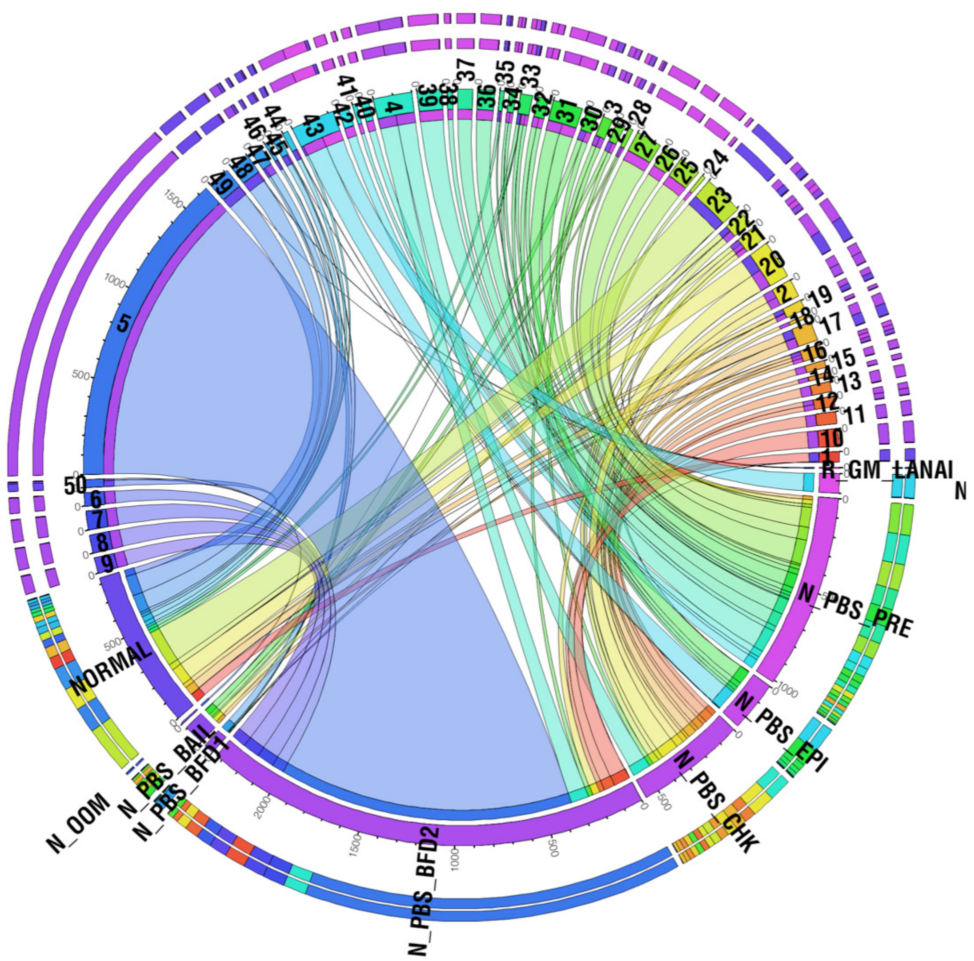
(c) BGL-Link

Figure C.1: Circos Plots for the BGL node categories. Each plot shows the plot for clustering as performed on all nodehours in each category.



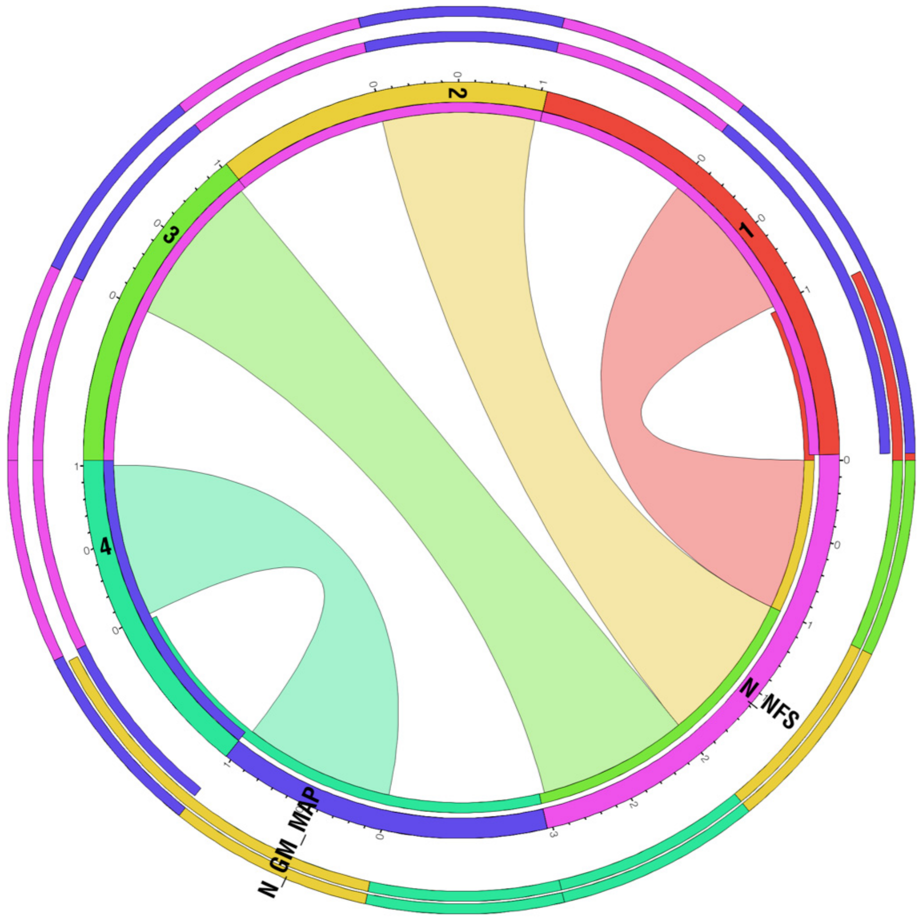
(d) BGL-Other

Figure C.1: Circos Plots for the BGL node categories. Each plot shows the plot for clustering as performed on all nodehours in each category.



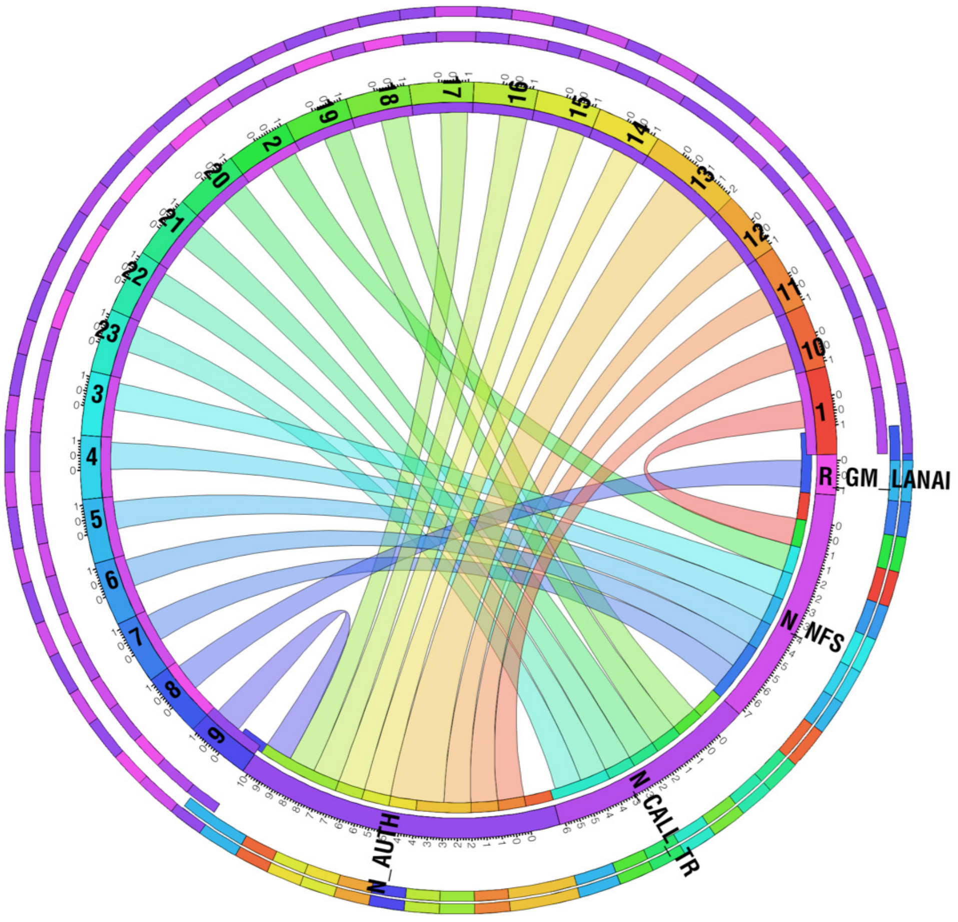
(e) Liberty-Compute

Figure C.2: Circos Plots for the Liberty node categories. The Liberty-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Liberty-Admin and Liberty-Other categories.



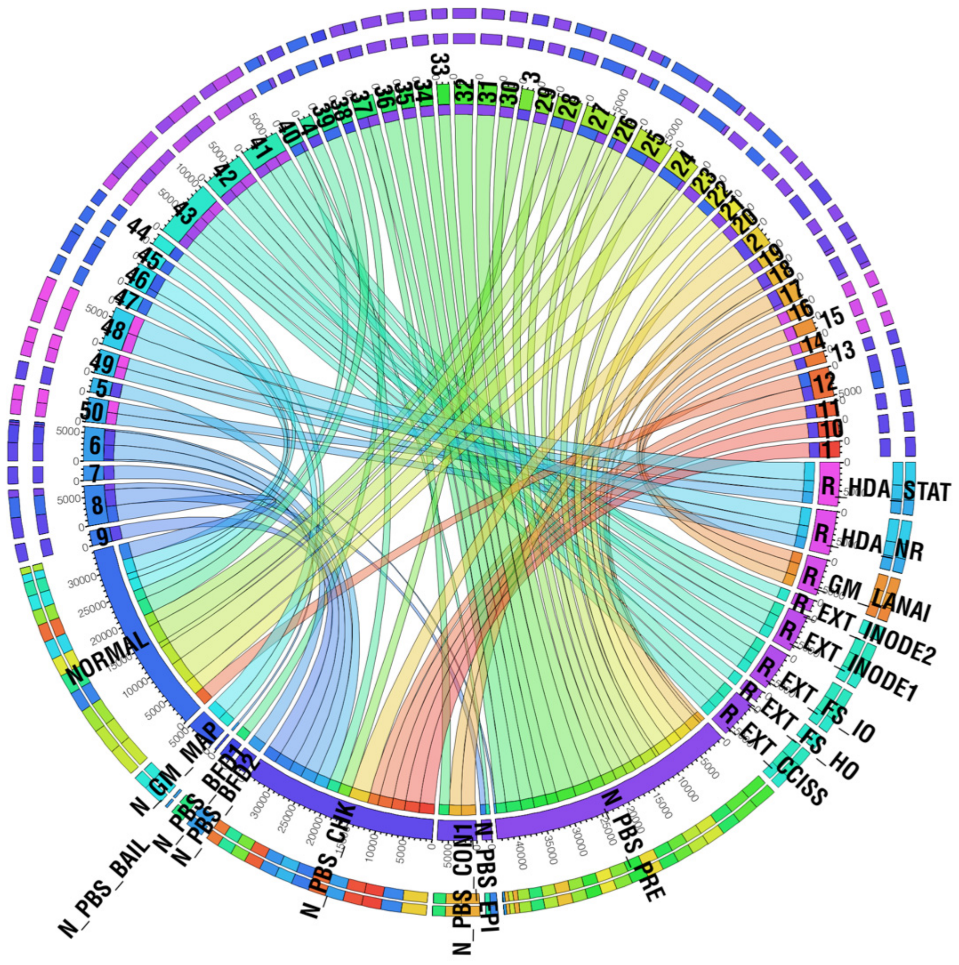
(f) Liberty-Admin Node1

Figure C.2: Circos Plots for the Liberty node categories. The Liberty-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Liberty-Admin and Liberty-Other categories.



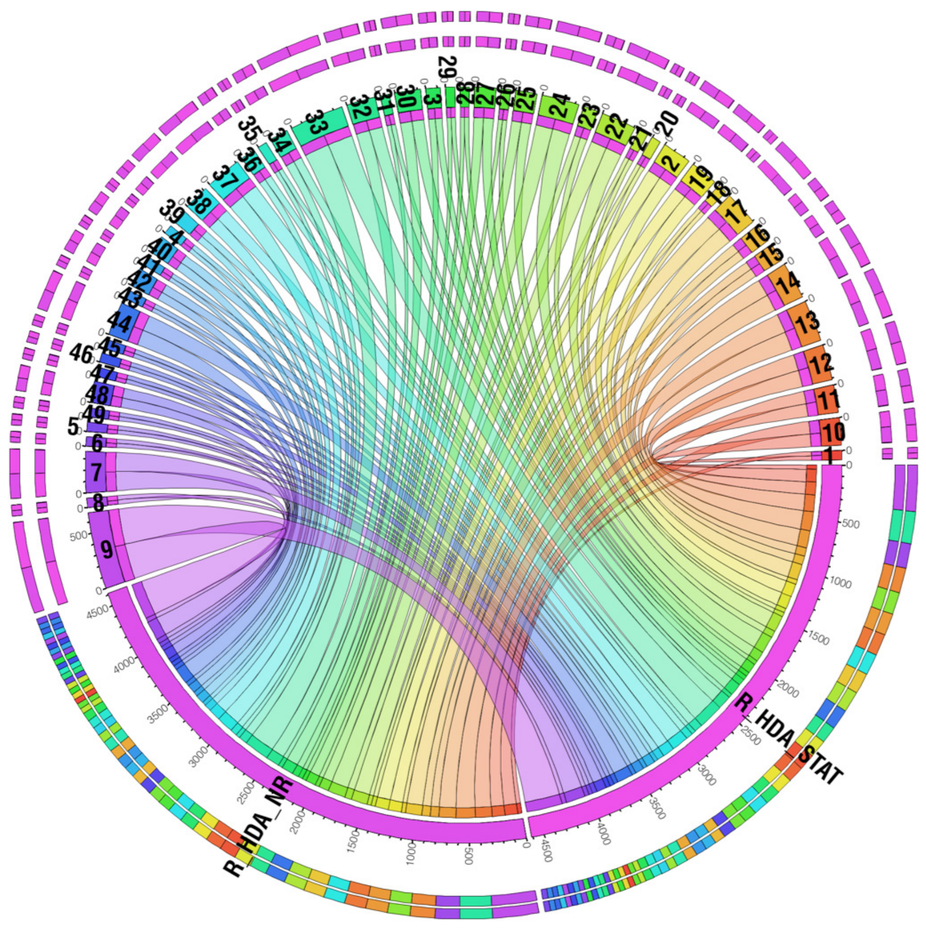
(g) Liberty-Other Node2

Figure C.2: Circos Plots for the Liberty node categories. The Liberty-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Liberty-Admin and Liberty-Other categories.



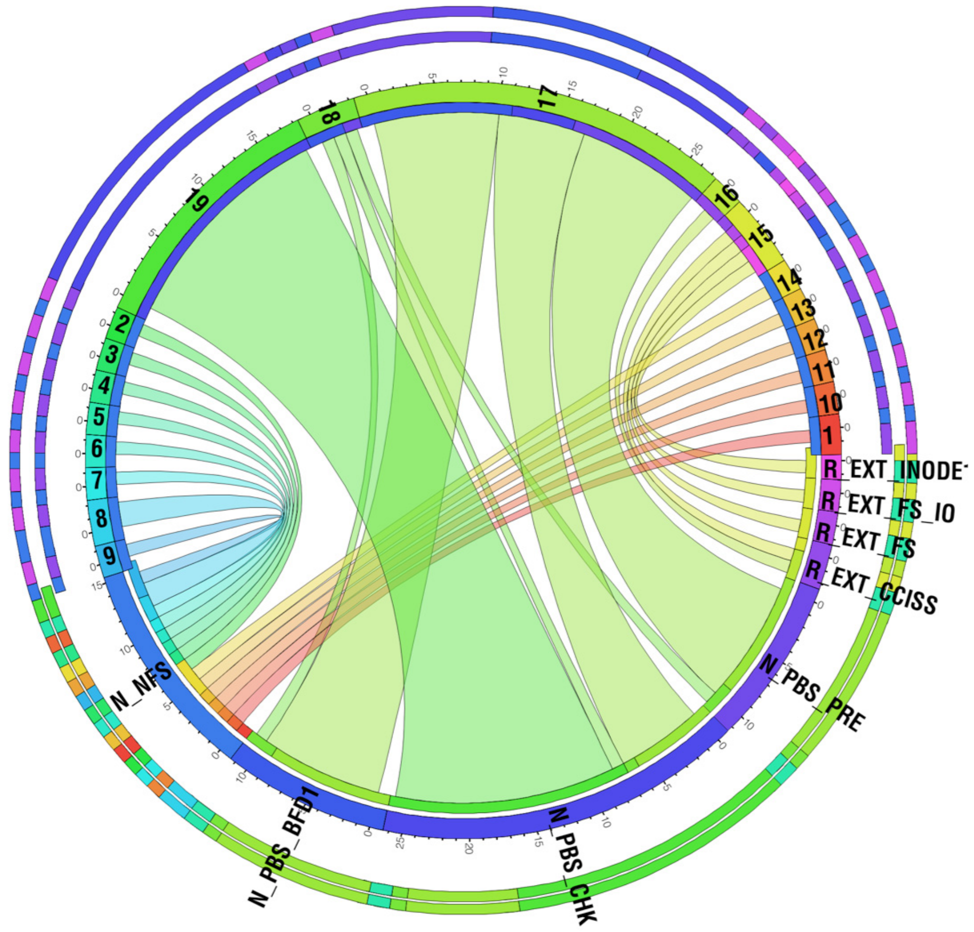
(h) Spirit-Compute

Figure C.3: Circos Plots for the Spirit node categories. The Spirit-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Spirit-Admin and Spirit-Other categories.



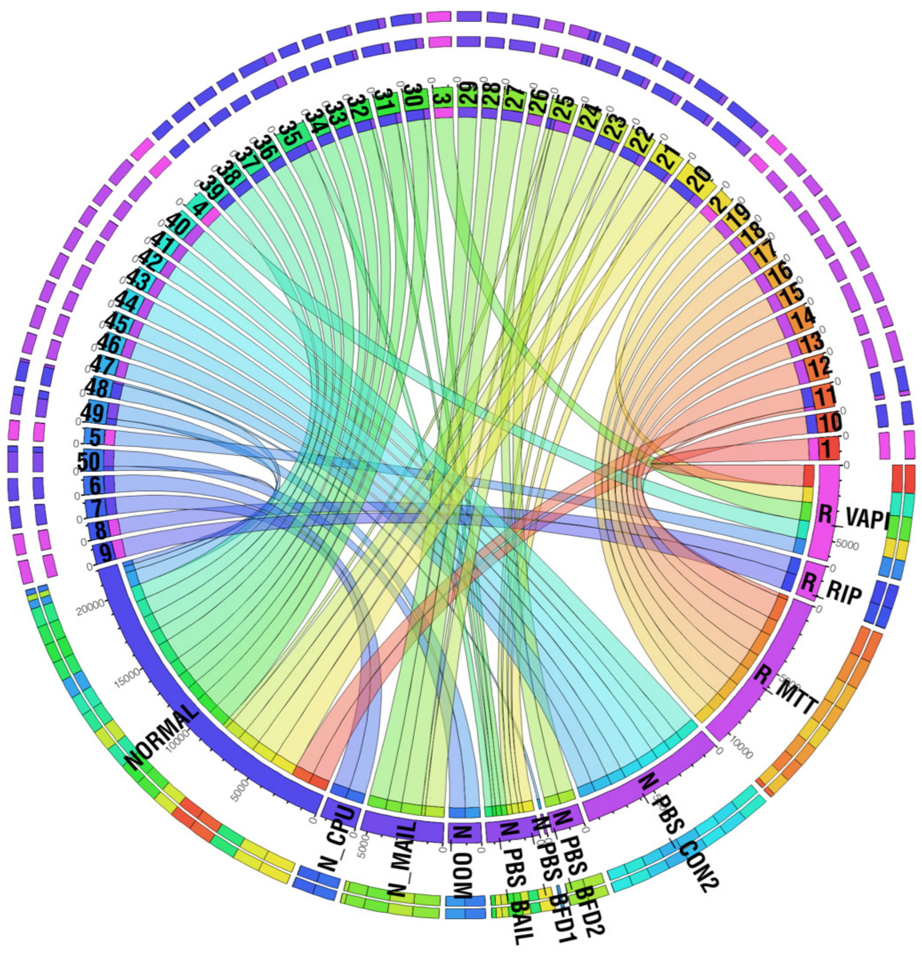
(i) Spirit-Admin Node0

Figure C.3: Circos Plots for the Spirit node categories. The Spirit-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Spirit-Admin and Spirit-Other categories.



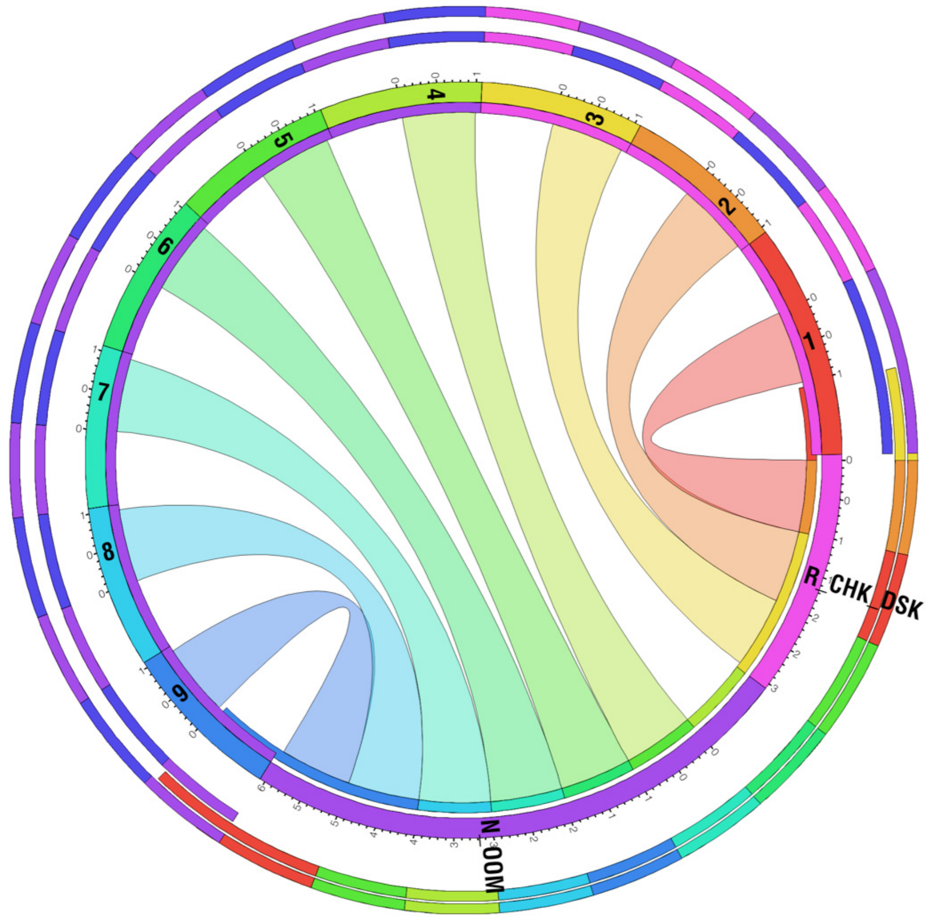
(j) Spirit-Other Node3

Figure C.3: Circos Plots for the Spirit node categories. The Spirit-Compute plot shows the plot for clustering as performed on all nodehours in this category, while only clusters formed from a single node are shown for Spirit-Admin and Spirit-Other categories.



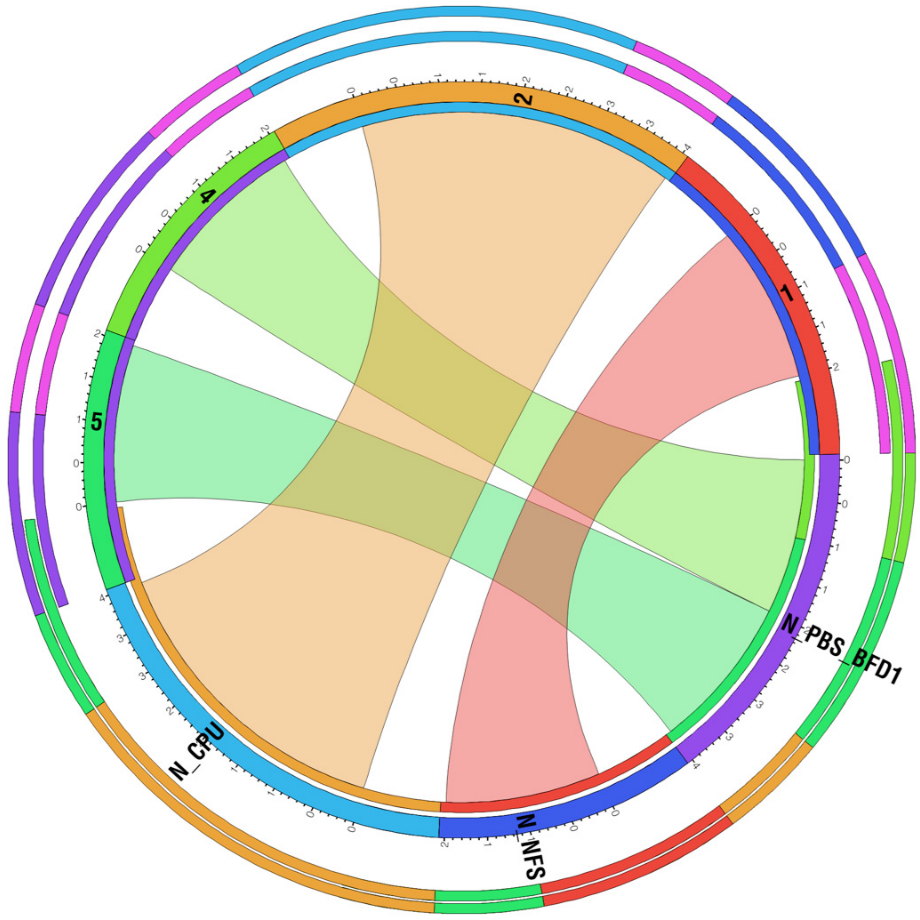
(k) Tbird-Compute

Figure C.4: Circos Plots for the Tbird node categories. The Tbird-Compute and Tbird-Other plots show plots for clustering as performed on all nodehours in each category, while only clusters formed from a single node are shown for Tbird-Admin category.



(l) Tbird-Admin Node18

Figure C.4: Circos Plots for the Tbird node categories. The Tbird-Compute and Tbird-Other plots show plots for clustering as performed on all nodehours in each category, while only clusters formed from a single node are shown for Tbird-Admin category.

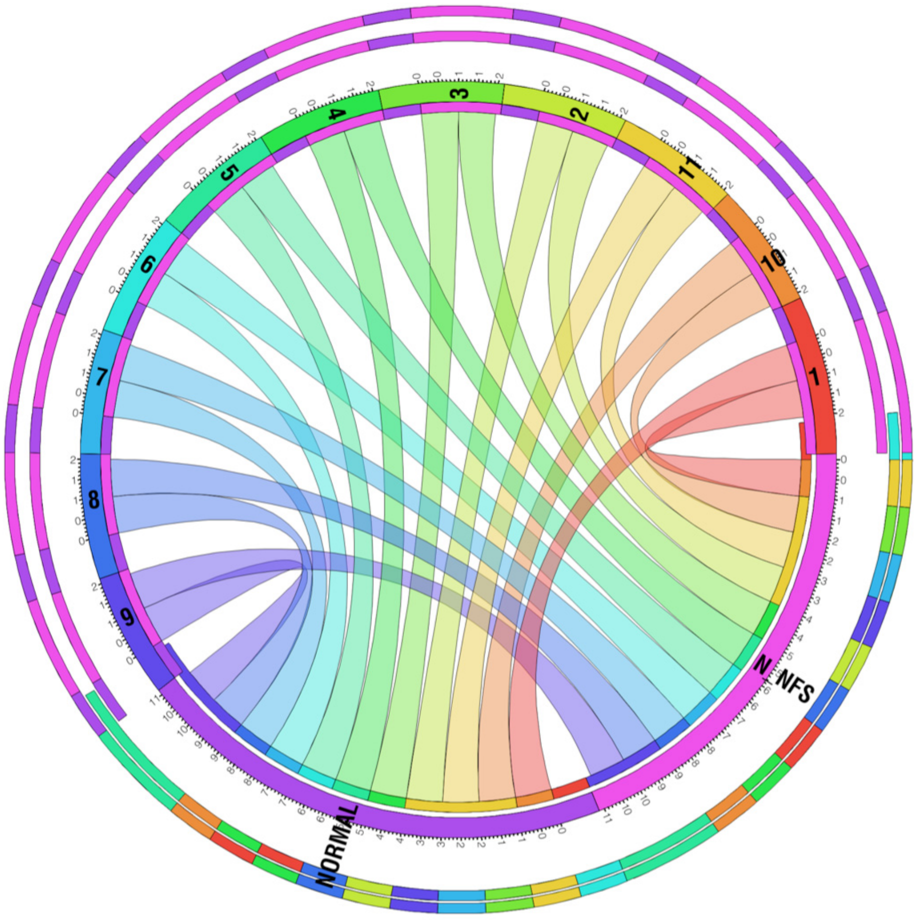


(m) Tbird-Other

Figure C.4: Circos Plots for the Tbird node categories. The Tbird-Compute and Tbird-Other plots show plots for clustering as performed on all nodehours in each category, while only clusters formed from a single node are shown for Tbird-Admin category.

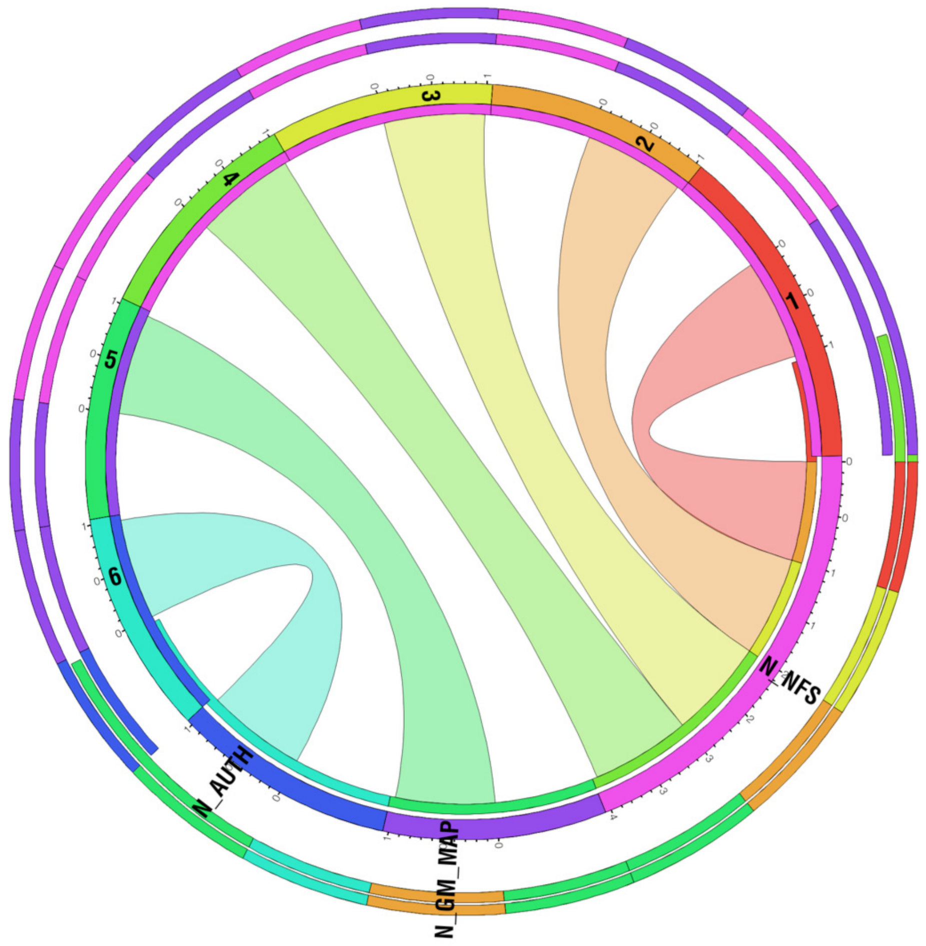
C.1.2 Other HPC Dissimilar Node Circos Plots

The Circos plots shown in Figs. C.5, C.6 and C.7 are for the nodes in Liberty-Admin, Liberty-Other, Spirit-Admin, Spirit-Other and Tbird-Admin. The nodes in this category were adjudged to be dissimilar from each other. Hence, anomaly detection was carried out on a node-by-node basis. The plots show how the clusters derived on each node map to the alert categories on the node.



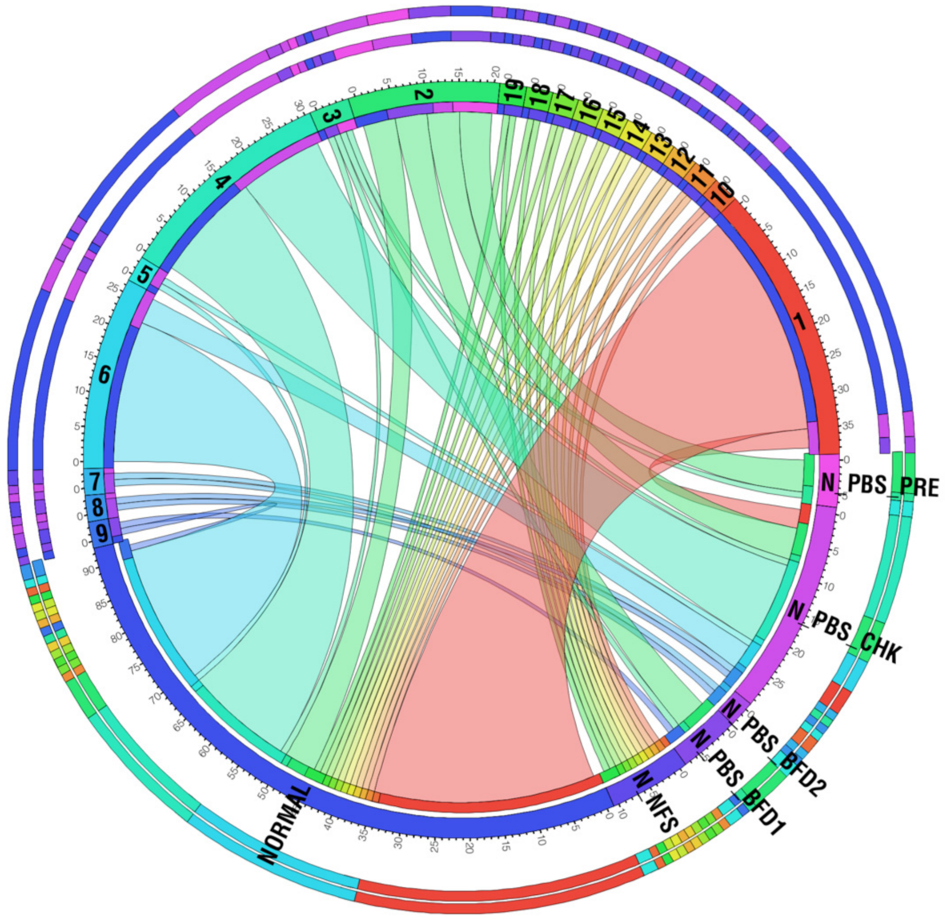
(n) Liberty-Admin Node0

Figure C.5: Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category.



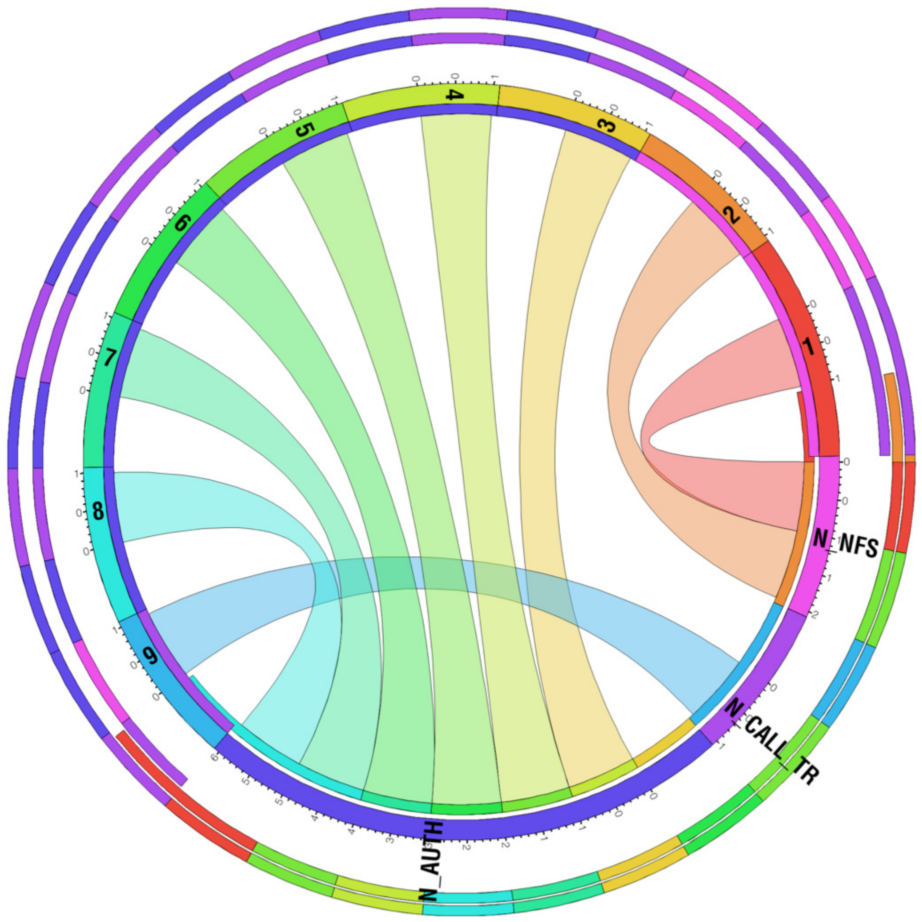
(o) Liberty-Other Node0

Figure C.5: Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category.



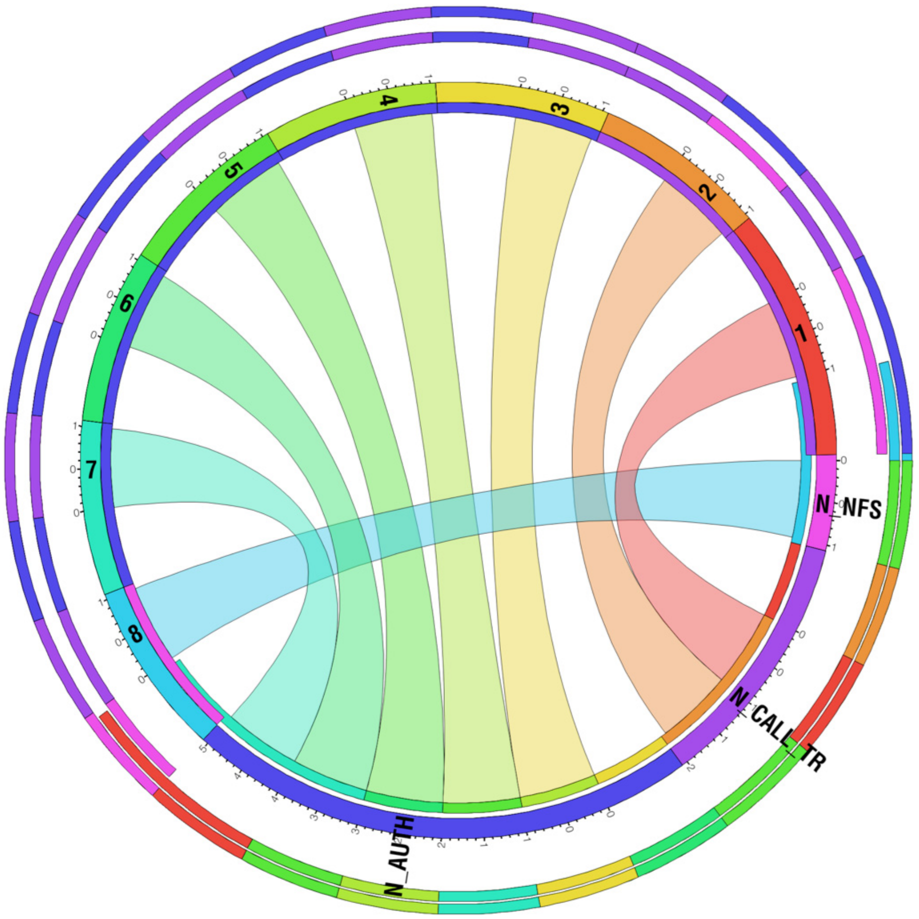
(p) Liberty-Other Node1

Figure C.5: Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category.



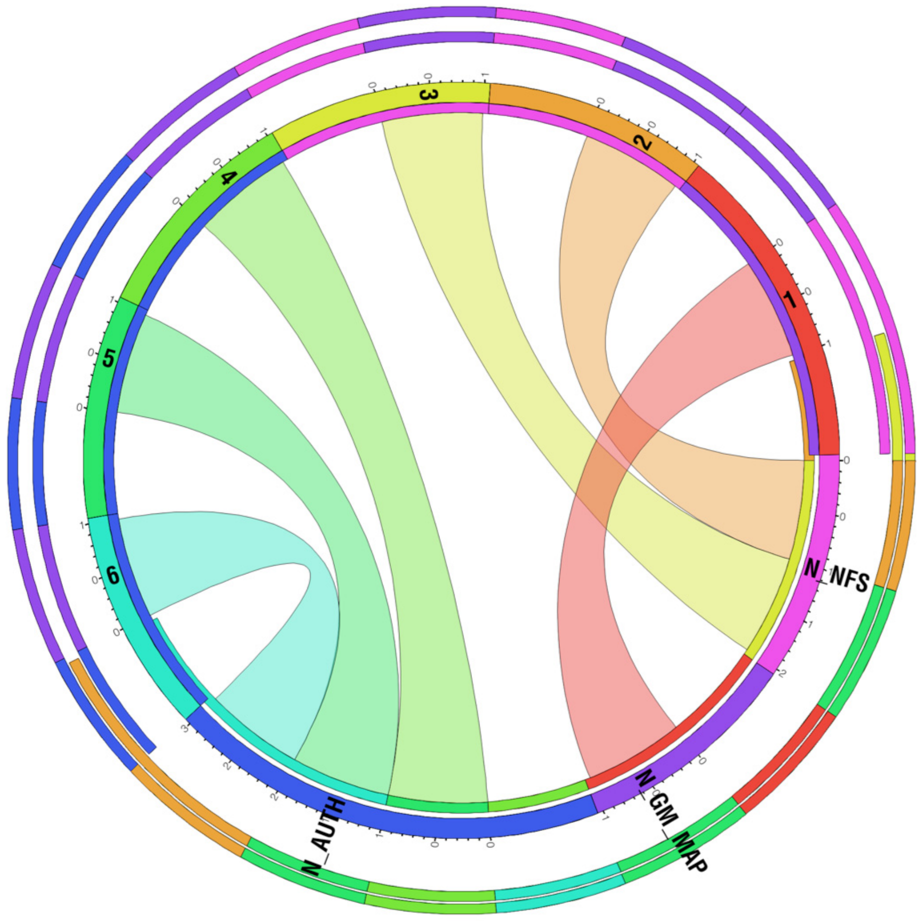
(q) Liberty-Other Node3

Figure C.5: Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category.



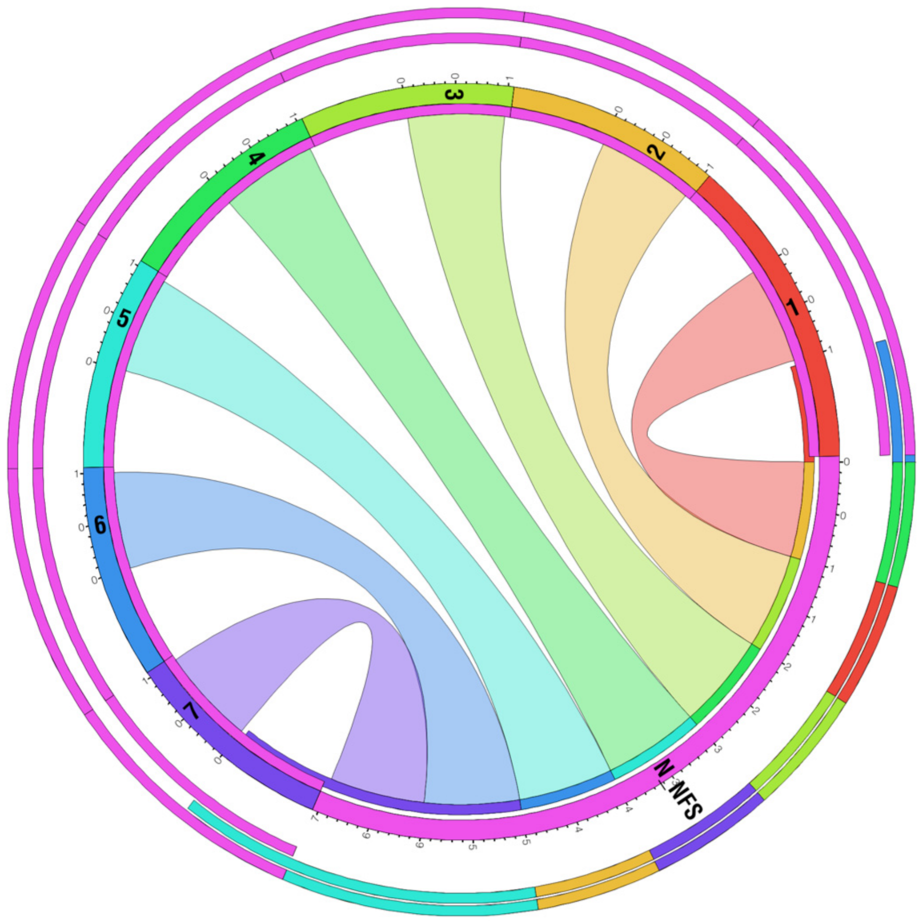
(r) Liberty-Other Node4

Figure C.5: Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category.



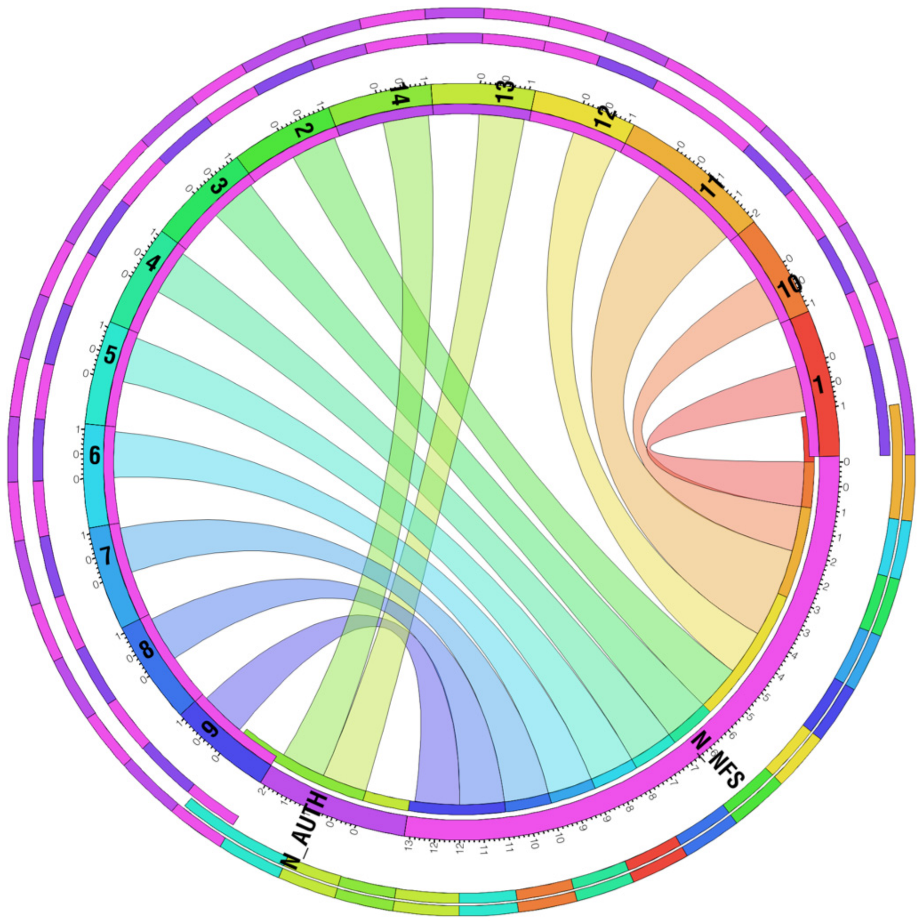
(s) Liberty-Other Node5

Figure C.5: Circos Plots for dissimilar nodes for the Liberty node categories. Each plot shows the plot for a single node in the category.



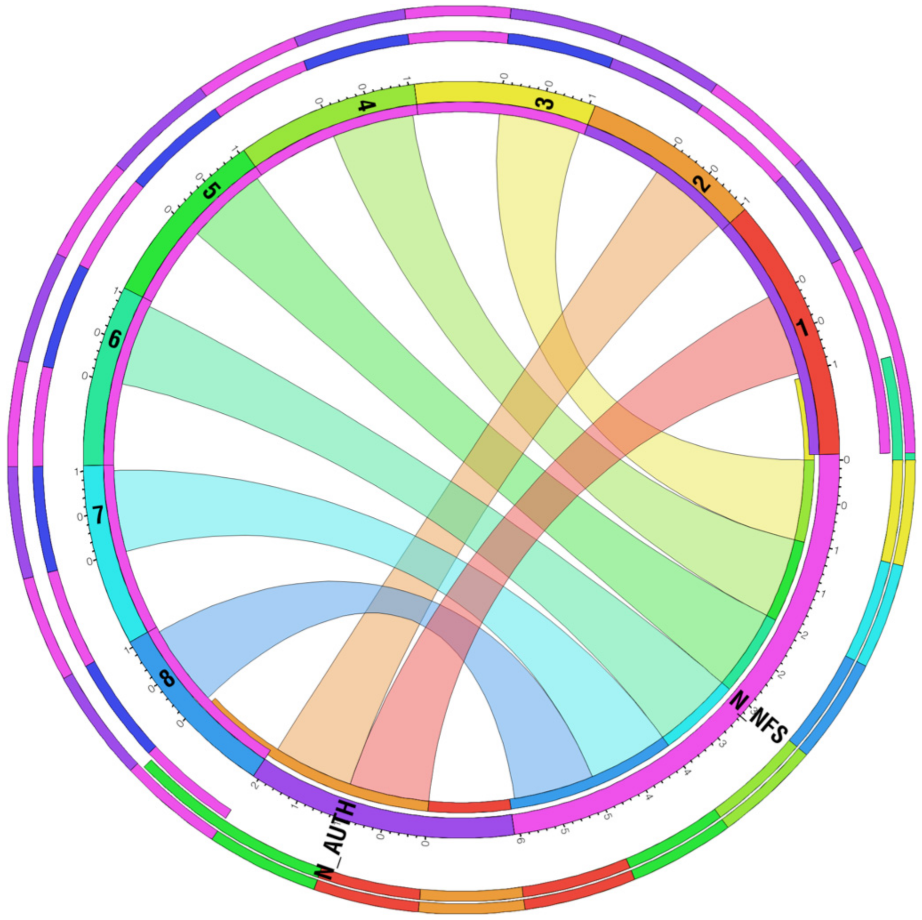
(a) Spirit-Admin Node1

Figure C.6: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



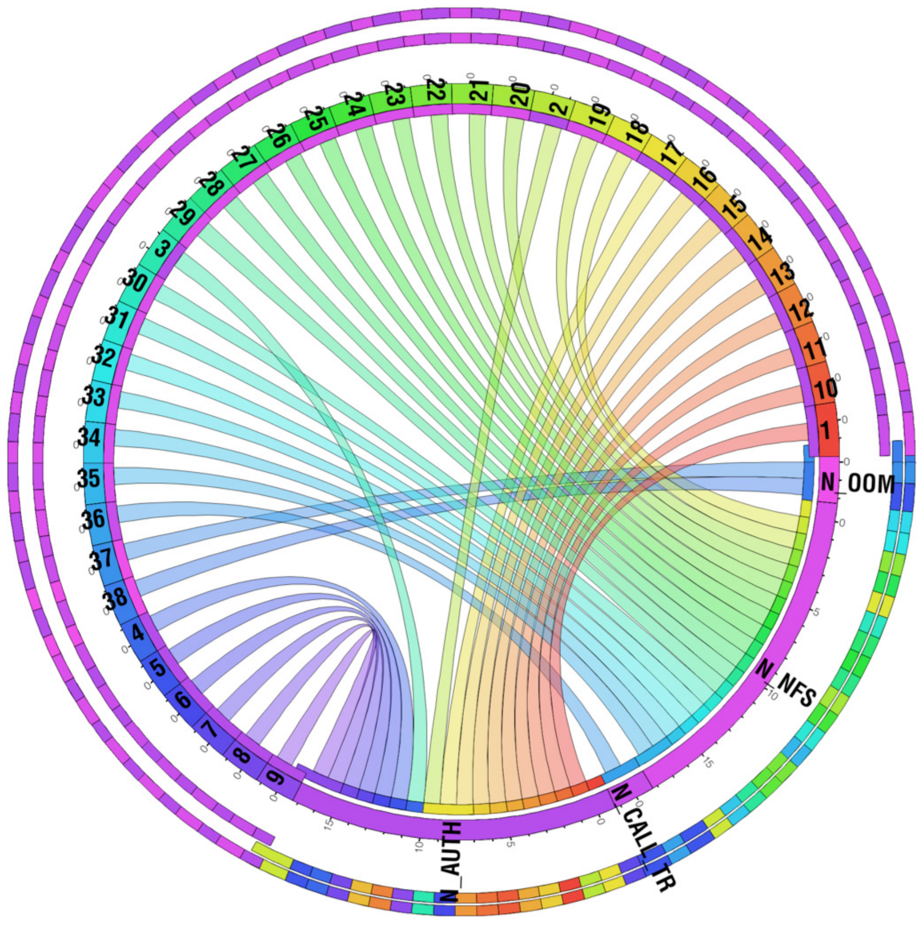
(b) Spirit-Other Node2

Figure C.6: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



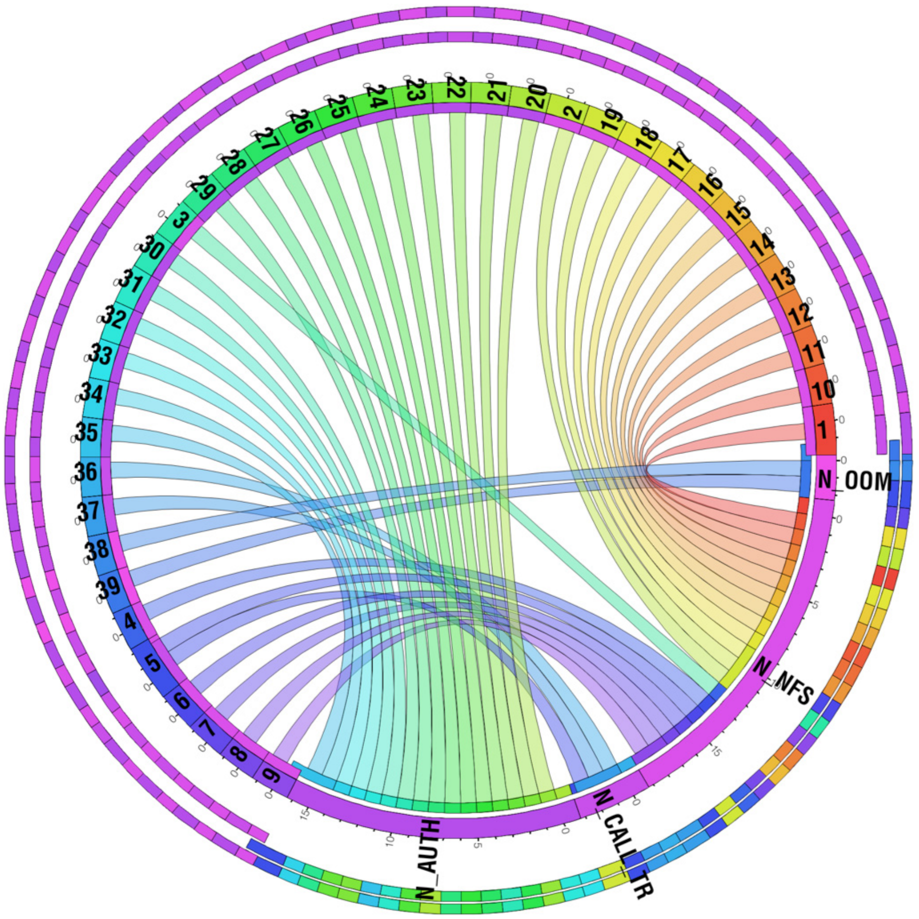
(c) Spirit-Other Node4

Figure C.6: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



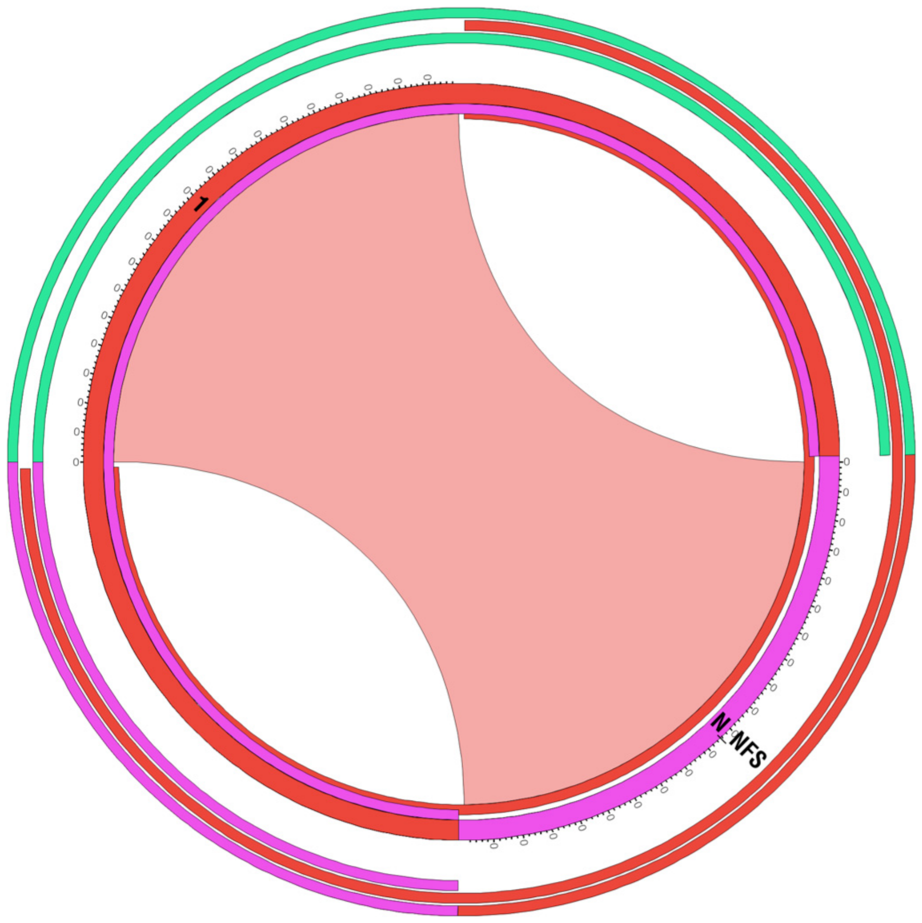
(d) Spirit-Other Node5

Figure C.6: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



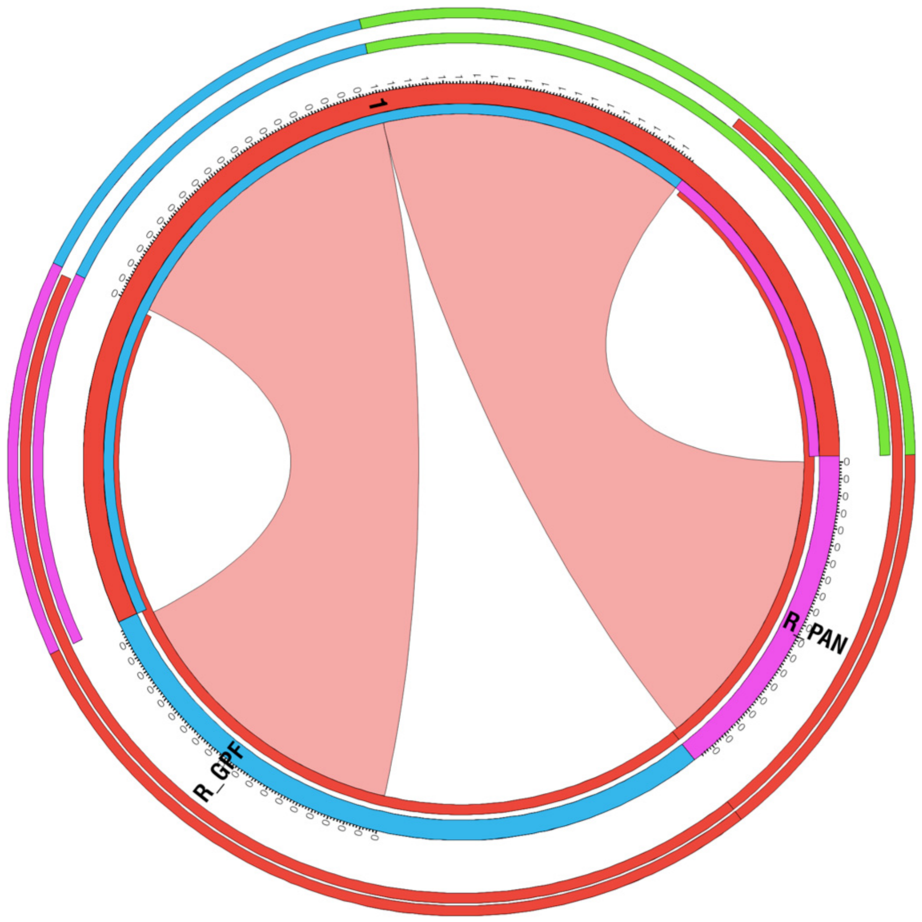
(e) Spirit-Other Node6

Figure C.6: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



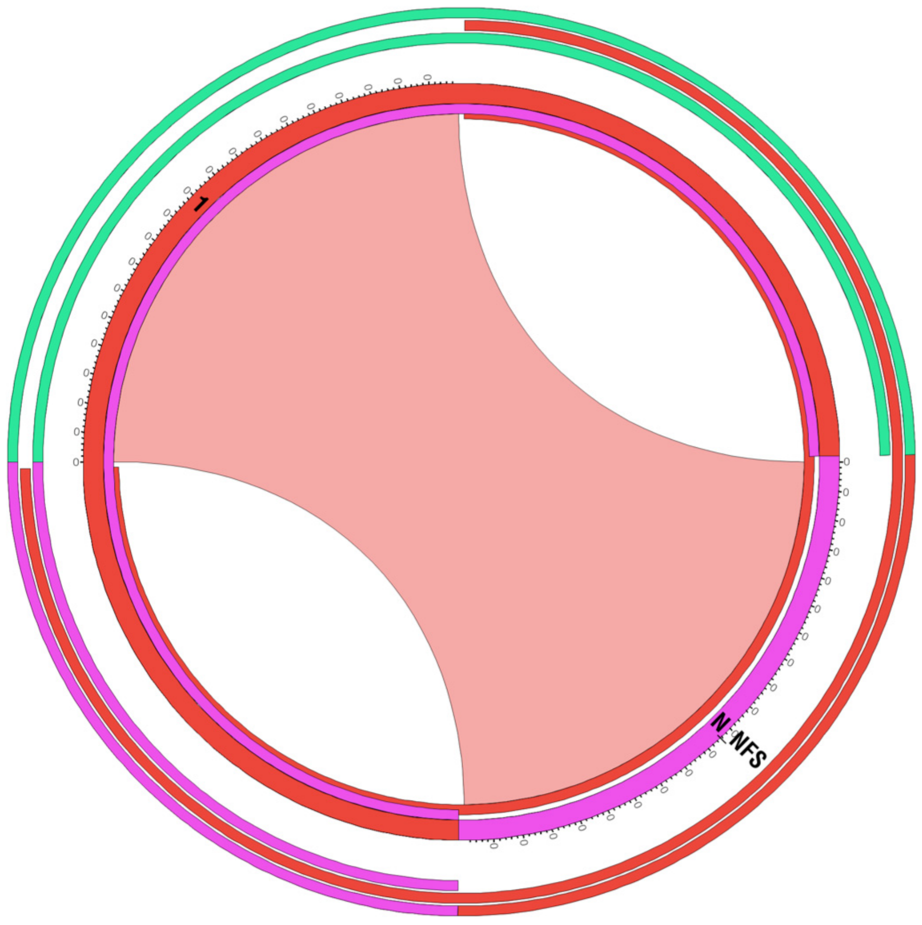
(a) Tbird-Admin Node2

Figure C.7: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



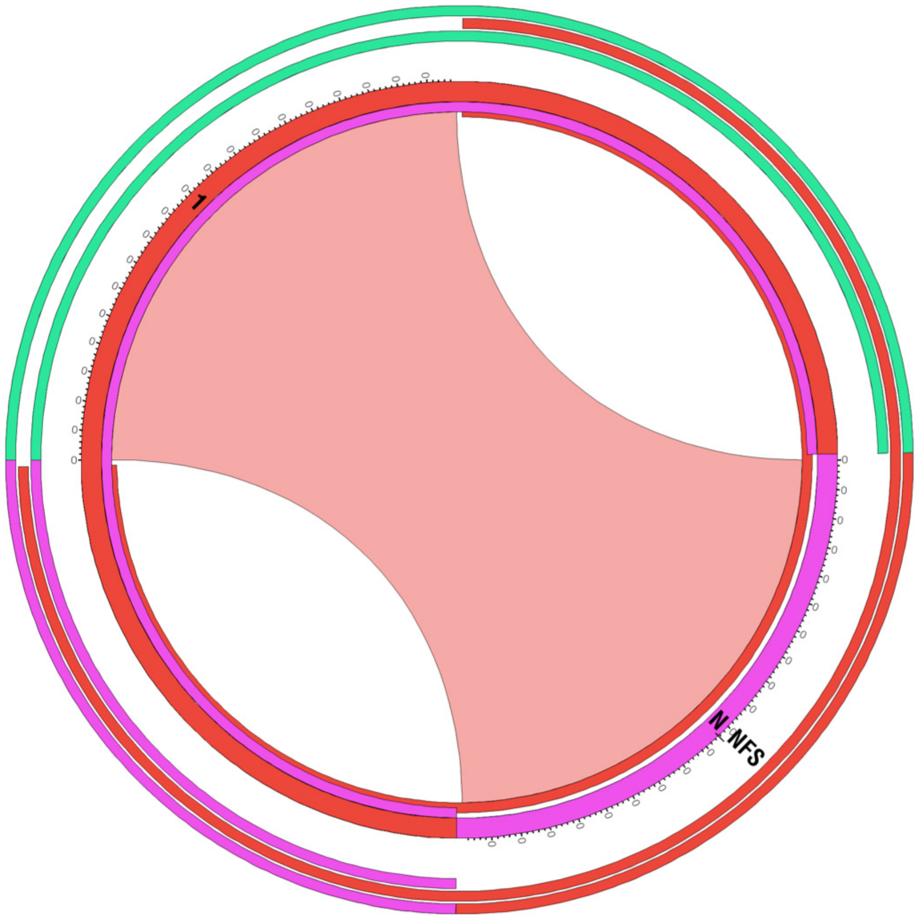
(b) Tbird-Admin Node5

Figure C.7: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



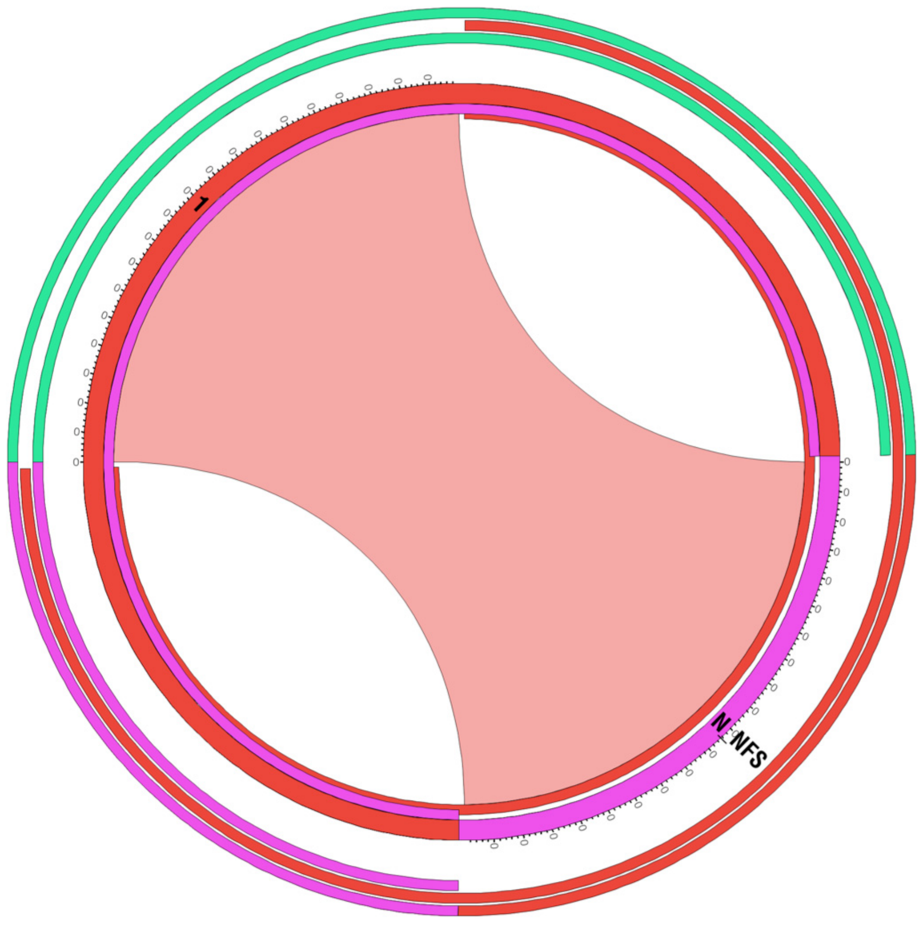
(c) Tbird-Admin Node8

Figure C.7: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



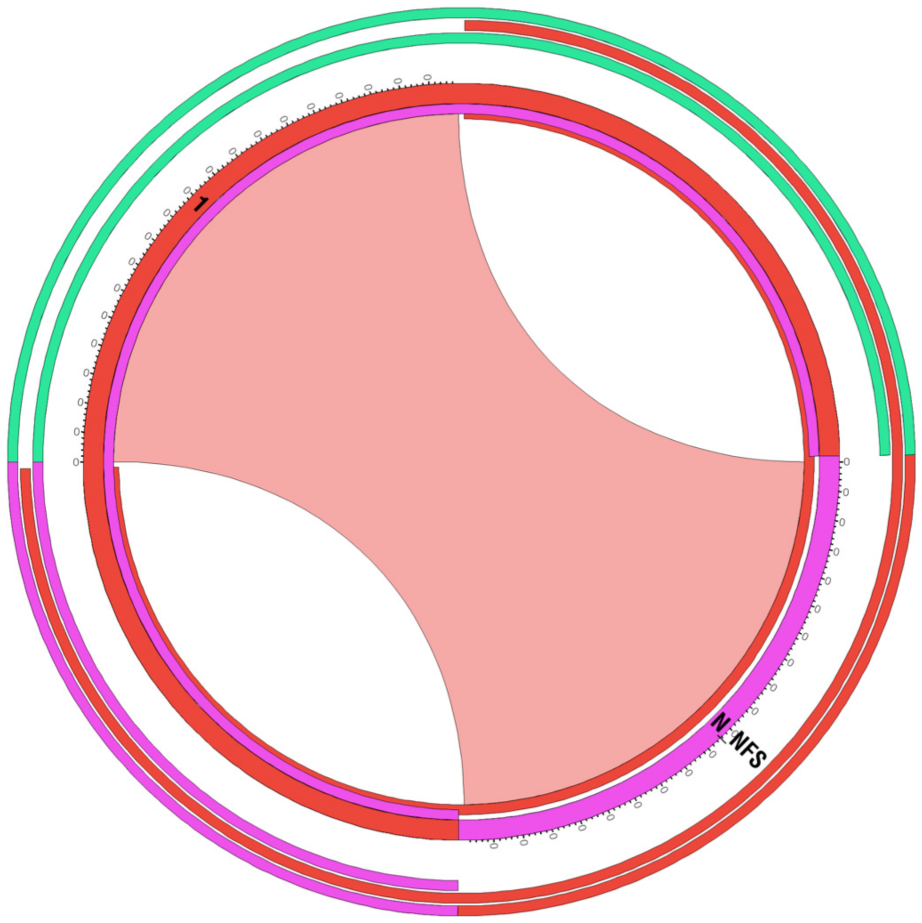
(d) Tbird-Admin Node13

Figure C.7: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



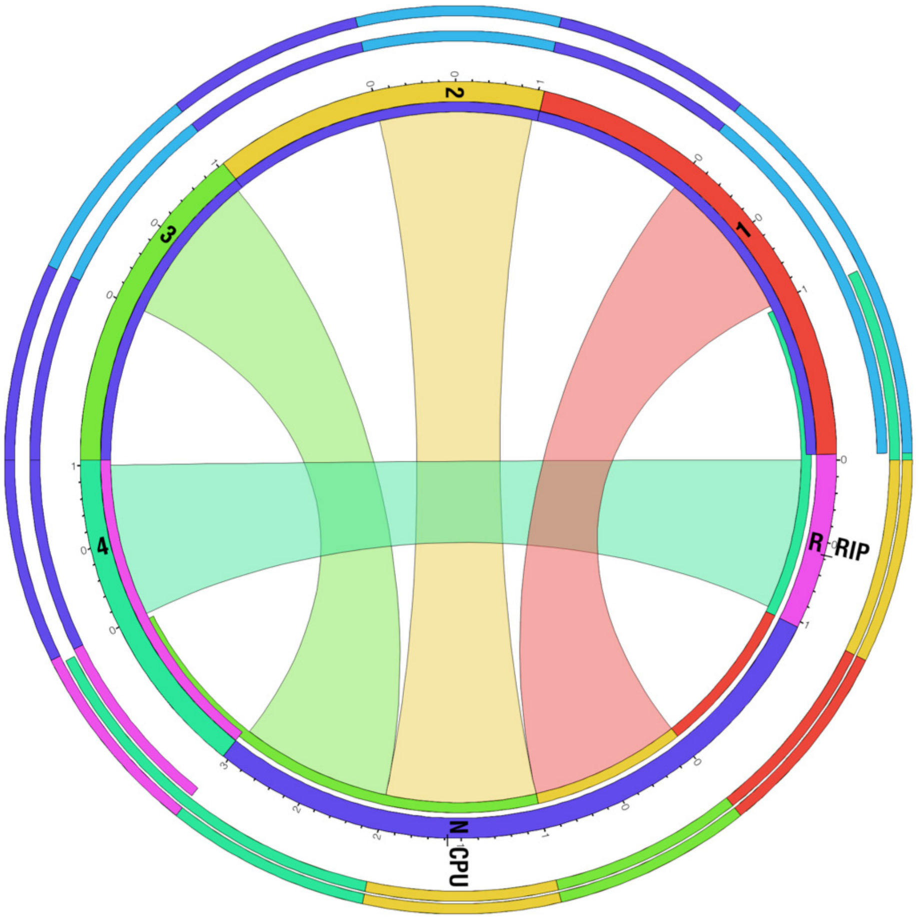
(e) Tbird-Admin Node14

Figure C.7: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



(f) Tbird-Admin Node15

Figure C.7: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.



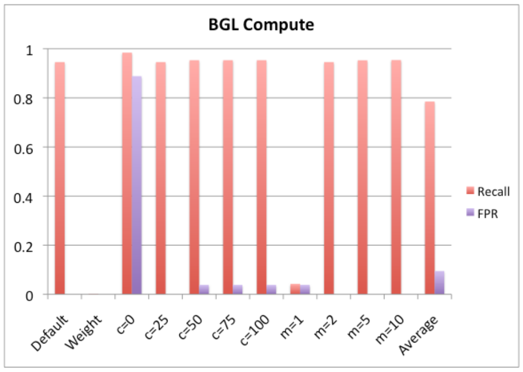
(g) Tbird-Admin Node19

Figure C.7: Circos Plots for dissimilar nodes for the Spirit node categories. Each plot shows the plot for a single node in the category.

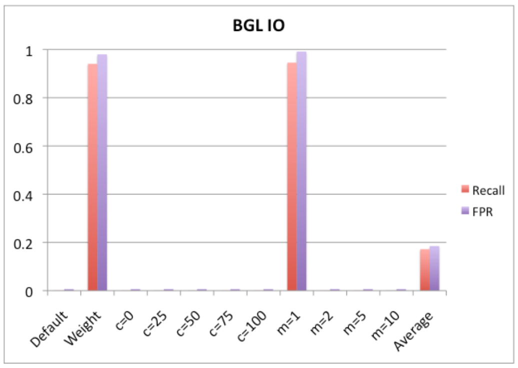
C.1.3 Detailed Results for C5.0 Evaluations

The Figs. C.8, C.9, C.10 and C.11 show the detailed results of running C5.0 on the cluster classification problem introduced in Chapter 6. In these figures, the results labelled *Default* are the same as those presented in Chapter 6, while the results labelled *Weight* are those obtained by weighting the *#Nodehours* feature during training. The results labelled *c=** and *m=** are obtained by varying the values of C5.0's *-c* and *-m* parameters. The meanings of these parameters are provided in Appendix A.

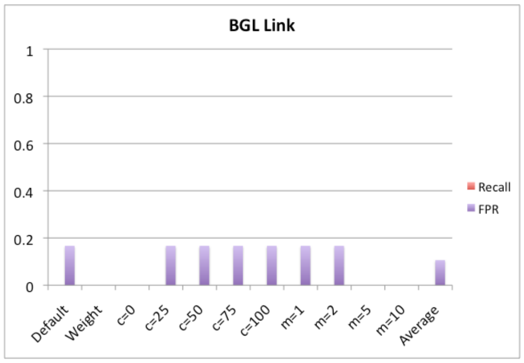
The results show that C5.0 achieves consistent Recall rates > 0.5 on the BGL-Compute and Liberty-Compute categories. Also, it achieves Recall rates > 0.5 in the BGL-IO category but these Recall rates are overshadowed by unacceptable FPRs. Indeed, acceptable FPRs were achieved in most cases except for the BGL-Compute, BGL-IO and BGL-Link categories. C5.0 was unable to produce a decent classifier for the Liberty-Admin, Tbird-Admin and Tbird-Other categories. The classifiers for these categories label everything as normal. Overall, no trend which indicates which parameter settings would improve classification results across all the datasets was observed.



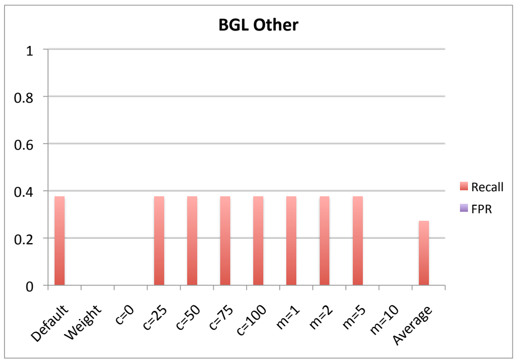
(h) BGL-Compute



(i) BGL-IO

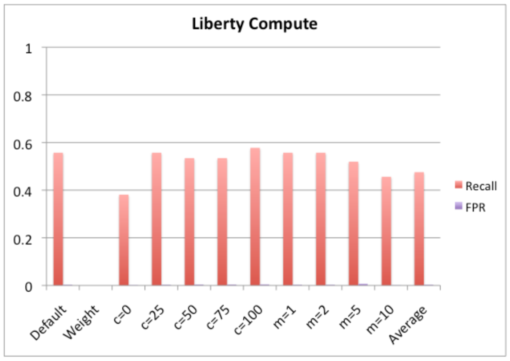


(j) BGL-Link

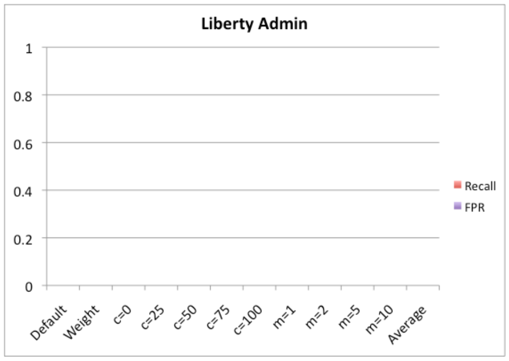


(k) BGL-Other

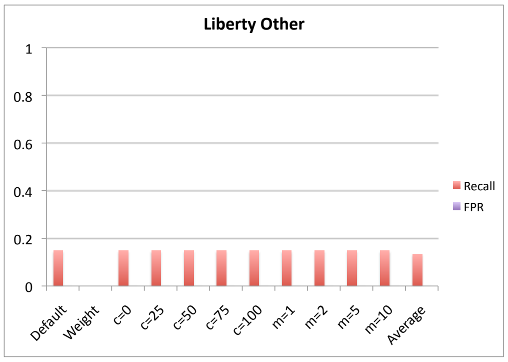
Figure C.8: Recall and FPR scores for the classification of anomalous and normal clusters for the BGL node categories using C5.0 classification rules.



(a) Liberty-Compute

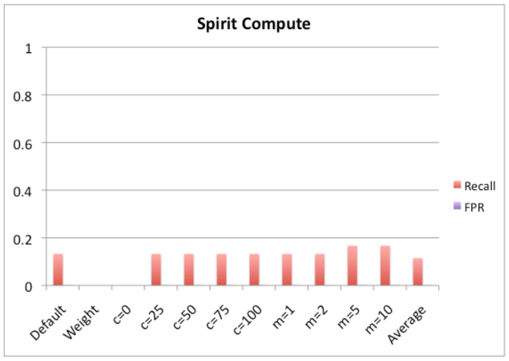


(b) Liberty-Admin

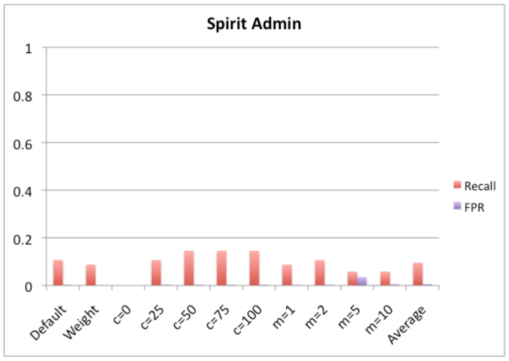


(c) Liberty-Other

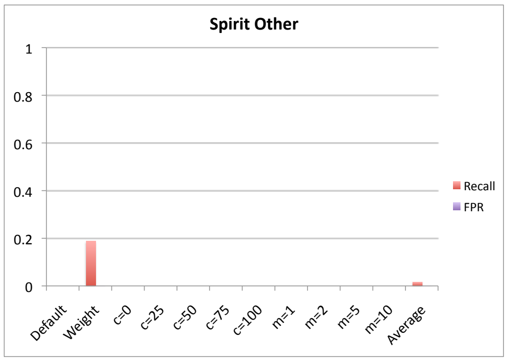
Figure C.9: Recall and FPR scores for the classification of anomalous and normal clusters for the Liberty node categories using C5.0 classification rules.



(a) Spirit-Compute

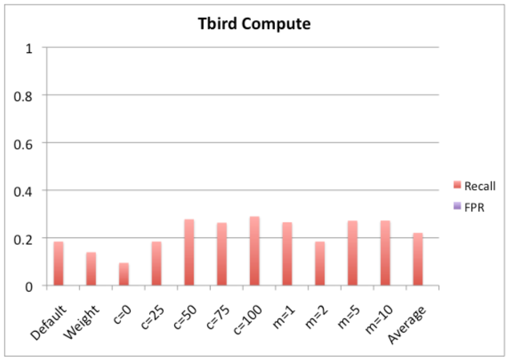


(b) Spirit-Admin

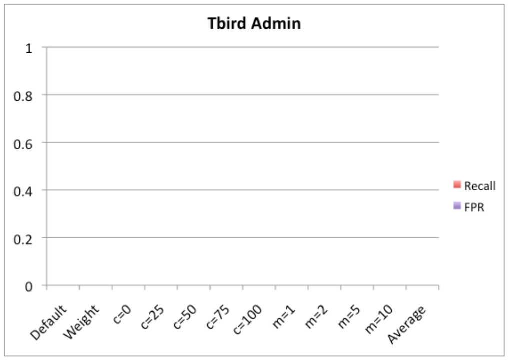


(c) Spirit-Other

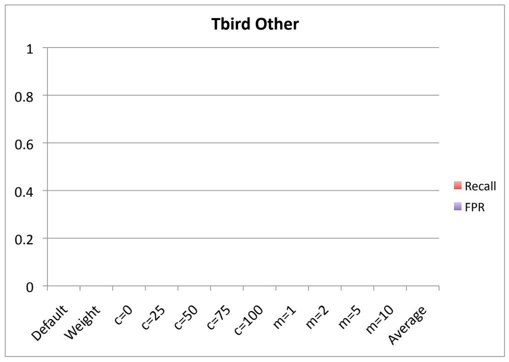
Figure C.10: Recall and FPR scores for the classification of anomalous and normal clusters for the Spirit node categories using C5.0 classification rules.



(a) Tbird-Compute



(b) Tbird-Admin



(c) Tbird-Other

Figure C.11: Recall and FPR scores for the classification of anomalous and normal clusters for the Tbird node categories using C5.0 classification rules.

Appendix D

D.1 Analysis of Variance (ANOVA) Tables

Detailed results of all the ANOVA tests performed as part of this thesis can be found here. All the ANOVA tests were carried out at 5% significance.

D.1.1 IPLoM Precision Performance

This section shows the results of the Anova test on the precision performance of IPLoM, SLCT and Loghound in Tables D.1, D.2, D.3, D.4, D.5, D.6 and D.7. The results show a statistically significant difference in all cases.

Table D.1: Anova Results Comparing Precision Performance on HPC Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	2.969	0.297	0.001		
Loghound	10	1.799	0.1799	0.001		
IPLoM (CD)	10	5.957	0.596	0.001		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.919	2	0.460	2133.742	1.9E-30	3.354
Within Groups	0.006	27	0.001			
Total	0.925	29				

Table D.2: Anova Results Comparing Precision Performance on Syslog Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	1.084	0.108	0.001		
Loghound	10	0.585	0.059	0.001		
IPLoM (CD)	10	1.540	0.154	0.001		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.046	2	0.023	30.961	1.03E-07	3.354
Within Groups	0.020	27	0.001			
Total	0.065	29				

Table D.3: Anova Results Comparing Precision Performance on Windows Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	3.127	0.313	0.001		
Loghound	10	0.739	0.074	5.89E-05		
IPLoM (CD)	10	6.290	0.629	0.001		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	1.550	2	0.775	1343.193	9.35E-28	3.354
Within Groups	0.016	27	0.001			
Total	1.566	29				

Table D.4: Anova Results Comparing Precision Performance on Access Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0	0	0		
Loghound	10	0.003	0.001	8.5E-07		
IPLoM (CD)	10	6.195	0.620	0.012		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	2.558	2	1.279	318.625	1.67E-19	3.354
Within Groups	0.108	27	0.004			
Total	2.666	29				

Table D.5: Anova Results Comparing Precision Performance on Error Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.368	0.037	0.001		
Loghound	10	0.203	0.020	2.63E-05		
IPLoM (CD)	10	9.015	0.901	0.012		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	5.082	2	2.541	603.307	3.91E-23	3.354
Within Groups	0.114	27	0.004			
Total	5.195	29				

Table D.6: Anova Results Comparing Precision Performance on System Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	1.111	0.111	8.56E-34		
Loghound	10	0.667	0.0667	0		
IPLoM (CD)	10	10	1	0		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	5.544	2	2.772	9.72E+33	0	3.354
Within Groups	7.7E-33	27	2.85E-34			
Total	5.544	29				

Table D.7: Anova Results Comparing Precision Performance on Rewrite Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.0958	0.009	1.52E-05		
Loghound	10	0.092	0.009	1.29E-05		
IPLoM (CD)	10	10	1	0		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	6.542	2	3.271	348431.5	2.76E-60	3.354
Within Groups	0.001	27	9.39E-06			
Total	6.543	29				

D.1.2 IPLoM Recall Performance

This section shows the results of the Anova test on the Recall performance of IPLoM, SLCT and Loghound in Tables D.8, D.9, D.10, D.11, D.12, D.13 and D.14. The results do not show a statistically significant difference in all cases: the HPC, Syslog and Windows files did not show a statistically significant difference.

Table D.8: Anova Results Comparing Recall Performance on HPC Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	1.466	0.1466	0.001		
Loghound	10	1.354	0.135	0.001		
IPLoM (CD)	10	2.498	0.250	0.001		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.079	2	0.040	35.346	2.89E-08	3.354
Within Groups	0.0304	27	0.001			
Total	0.110	29				

Table D.9: Anova Results Comparing Recall Performance on Syslog Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.733	0.073	0.002		
Loghound	10	0.733	0.073	0.002		
IPLoM (CD)	10	0.833	0.083	0.003		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.001	2	0.001	0.130	0.878	3.354
Within Groups	0.069	27	0.002			
Total	0.070	29				

Table D.10: Anova Results Comparing Recall Performance on Windows Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.826	0.083	0.001		
Loghound	10	1.056	0.106	0.001		
IPLoM (CD)	10	1.286	0.129	0.001		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.011	2	0.005	3.882	0.033	3.354
Within Groups	0.037	27	0.001			
Total	0.047	29				

Table D.11: Anova Results Comparing Recall Performance on Access Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0	0	0		
Loghound	10	0.067	0.007	0.001		
IPLoM (CD)	10	1.786	0.179	0.001		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.205	2	0.102	165.129	7.21E-16	3.354
Within Groups	0.017	27	0.001			
Total	0.222	29				

Table D.12: Anova Results Comparing Recall Performance on Error Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.114	0.011	8.43E-05		
Loghound	10	0.114	0.011	8.43E-05		
IPLoM (CD)	10	0.415	0.0416	0.001		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.006	2	0.003	21.087	3.05E-06	3.354
Within Groups	0.004	27	0.001			
Total	0.010	29				

Table D.13: Anova Results Comparing Recall Performance on System Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.625	0.063	0		
Loghound	10	0.455	0.0455	5.35E-35		
IPLoM (CD)	10	5.556	0.556	1.37E-32		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	1.679	2	0.839	1.83E+32	0	3.354
Within Groups	1.24E-31	27	4.58E-33			
Total	1.679	29				

Table D.14: Anova Results Comparing Recall Performance on Rewrite Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.160	0.016	3.12E-05		
Loghound	10	0.155	0.015	2.73E-05		
IPLoM (CD)	10	3	0.3	3.42E-33		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.539	2	0.269	13812.74	2.29E-41	3.354
Within Groups	0.001	27	1.95E-05			
Total	0.539	29				

D.1.3 IPLoM F-Measure Performance

This section shows the results of the Anova test on the F-Measure performance of IPLoM, SLCT and Loghound in Tables D.15, D.16, D.17, D.18, D.19, D.20 and D.21.

Table D.15: Anova Results Comparing F-Measure Performance on HPC Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	1.466	0.147	0.001		
Loghound	10	1.354	0.135	0.001		
IPLoM (CD)	10	2.498	0.250	0.001		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.079	2	0.040	35.346	2.88629E-08	3.354
Within Groups	0.0304	27	0.001			
Total	0.110	29				

Table D.16: Anova Results Comparing F-Measure Performance on Syslog Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.815	0.0815	0.001		
Loghound	10	0.620	0.0620	0.001		
IPLoM (CD)	10	0.964	0.0964	0.003		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.006	2	0.003	1.631	0.215	3.354
Within Groups	0.050	27	0.001			
Total	0.055	29				

Table D.17: Anova Results Comparing F-Measure Performance on Windows Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	1.269	0.127	0.001		
Loghound	10	0.857	0.086	0.001		
IPLoM (CD)	10	2.110	0.211	0.003		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.082	2	0.041	26.547	4.215E-07	3.354
Within Groups	0.0415	27	0.002			
Total	0.123	29				

Table D.18: Anova Results Comparing F-Measure Performance on Access Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0	0	0		
Loghound	10	0.006	0.001	3.121E-06		
IPLoM (CD)	10	2.732	0.273	0.002		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.496	2	0.248	366.309	2.725E-20	3.354
Within Groups	0.018	27	0.001			
Total	0.515	29				

Table D.19: Anova Results Comparing F-Measure Performance on Error Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.148	0.015	2.792E-05		
Loghound	10	0.127	0.013	1.657E-05		
IPLoM (CD)	10	0.787	0.079	0.001		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.028	2	0.014	51.348	6.294E-10	3.354
Within Groups	0.007	27	0.001			
Total	0.036	29				

Table D.20: Anova Results Comparing F-Measure Performance on System Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.8	0.08	2.140E-34		
Loghound	10	0.541	0.054	0		
IPLoM (CD)	10	7.143	0.714	0		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	2.796	2	1.398	1.960E+34	0	3.354
Within Groups	1.93E-33	27	7.13E-35			
Total	2.796	29				

Table D.21: Anova Results Comparing F-Measure Performance on Rewrite Log.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
SLCT	10	0.120	0.012	2.129E-05		
Loghound	10	0.115	0.012	1.832E-05		
IPLoM (CD)	10	4.615	0.462	3.423E-33		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	1.349	2	0.674	51076.721	4.977E-49	3.354
Within Groups	0.001	27	1.32E-05			
Total	1.349	29				

D.1.4 STAD and C5.0 Rule-set ANOVA Results

The tables in this section show the results for the ANOVA tests performed on the Recall and FPR performance of cluster identification by the manual rule-set, C5.0 rule-set and *NIPlus*. The results of the test between the C5.0 and manual set performances are in Tables D.22 and D.23. The results of the test between the best case of manually setting manual rule-set parameters vs. setting them automatically are in Tables D.24 and D.25, while, the results of the test between the manual rule-set and *NIPlus* performances are in Tables D.26 and D.27.

Table D.22: Anova Results Comparing Recall Performance of C5.0 and Rule-Set on Cluster Classification

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
Recall-Rules	14	10.842	0.774	0.038		
Recall-C50	14	2.638	0.188	0.074		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	2.404	1	2.404	42.641	6.36E-07	4.225
Within Groups	1.466	26	0.056			
Total	3.870	27				

Table D.23: Anova Results Comparing FPR Performance of C5.0 and Rule-Set on Cluster Classification

SUMMARY							
Groups	*Count*	*Sum*	*Average*	*Variance*			
FPR-Rules	14	0.969	0.069	0.002			
FPR-C50	14	0.196	0.014	0.001			
ANOVA							
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*	
Between Groups	0.021	1	0.021	10.529	0.003	4.225	
Within Groups	0.052	26	0.002				
Total	0.0740	27					

Table D.24: Anova Results Comparing Recall Performance of Rule-Set Cluster Classifier when parameters are set automatically and best case when parameters are set manually.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
Recall-Best Case	13	10.124	0.779	0.037		
Recall-Autoset	13	10.068	0.774	0.041		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.0001	1	0.0001	0.003	0.957	4.260
Within Groups	0.947	24	0.039			
Total	0.947238314	25				

Table D.25: Anova Results Comparing FPR Performance of Rule-Set Cluster Classifier when parameters are set automatically and best case when parameters are set manually.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
FPR -Best Case	13	0.703	0.054	0.001		
FPR-Autoset	13	0.900	0.069	0.002		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.001	1	0.001	0.817	0.375	4.26
Within Groups	0.044	24	0.002			
Total	0.04524635	25				

Table D.26: Anova Results Comparing Recall Performance of *NIPlus* and best case of Rule-Set Cluster Classifier .

SUMMARY							
Groups	*Count*	*Sum*	*Average*	*Variance*			
FPR (NIPlus)	13	3.357	0.258	0.090			
FPR -Best Case	13	0.703	0.054	0.001			
ANOVA							
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*	
Between Groups	0.271	1	0.271	5.934	0.023	4.260	
Within Groups	1.096	24	0.046				
Total	1.367	25					

Table D.27: Anova Results Comparing FPR Performance of *NIPlus* and Rule-Set Cluster Classifier when parameters are set automatically.

SUMMARY							
Groups	*Count*	*Sum*	*Average*	*Variance*			
FPR (NIPlus)	13	3.357	0.258	0.089			
FPR-Autoset	13	0.899	0.069	0.002			
ANOVA							
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*	
Between Groups	0.232	1	0.232	5.036	0.034	4.260	
Within Groups	1.107	24	0.046				
Total	1.339	25					

D.1.5 Online detection ANOVA Results

The tables in this section show the results for the ANOVA tests performed on the Recall and FPR performance of online alert detection on the HPC, distributed and cloud logs using different training file split points. The results of the tests on the HPC logs are in Tables D.28 and D.29. The results of the tests on the distributed logs are in Tables D.30 and D.31, whlie, the results of the tests on the cloud logs are in Tables D.32 and D.33.

Table D.28: Anova Results Comparing Recall Performance with different training file split points on the HPC Logs.

SUMMARY							
Groups	*Count*	*Sum*	*Average*	*Variance*			
0.1	13	11.324	0.871	0.014			
0.2	13	11.497	0.884	0.015			
0.3	13	11.693	0.899	0.011			
0.4	12	10.436	0.870	0.014			
0.5	12	10.430	0.869	0.013			
ANOVA							
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*	
Between Groups	0.009	4	0.002	0.162	0.957	2.530	
Within Groups	0.790	58	0.014				
Total	0.799157505	62					

Table D.29: Anova Results Comparing FPR Performance with different training file split points on the HPC Logs.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
0.1	13	3.986E-05	3.066E-06	1.223E-10		
0.2	13	2.335E-06	1.796E-07	4.195E-13		
0.3	13	2.335E-06	1.796E-07	4.194E-13		
0.4	13	2.335E-06	1.796E-07	4.194E-13		
0.5	13	2.335E-06	1.796E-07	4.194E-13		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	8.66801E-11	4	2.167E-11	0.874	0.485	2.525
Within Groups	1.487E-09	60	2.478E-11			
Total	1.574E-09	64				

Table D.30: Anova Results Comparing DR Performance with different training file split points on the Distributed Logs.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
0.1	4	3.368	0.842	0.034		
0.2	4	3.365	0.841	0.034		
0.3	4	3.667	0.917	0.028		
0.4	4	3.5	0.875	0.063		
0.5	4	3	0.75	0.083		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.060	4	0.015	0.31	0.866	3.056
Within Groups	0.728	15	0.049			
Total	0.788	19				

Table D.31: Anova Results Comparing FPR Performance with different training file split points on the Distributed Logs.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
0.1	5	0	0	0		
0.2	5	0	0	0		
0.3	5	0	0	0		
0.4	5	0	0	0		
0.5	5	0	0	0		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0	4	0	65535	NA	2.866
Within Groups	0	20	0			
Total	0	24				

Table D.32: Anova Results Comparing DR Performance with different training file split points on the Cloud Logs.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
0.1	3	2.586	0.862	0.028		
0.2	3	2.556	0.852	0.023		
0.3	3	2.556	0.852	0.032		
0.4	3	2.432	0.811	0.058		
0.5	3	2.375	0.792	0.068		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.011	4	0.003	0.068	0.990	3.478
Within Groups	0.418	10	0.042			
Total	0.429	14				

Table D.33: Anova Results Comparing FPR Performance with different training file split points on the Cloud Logs.

SUMMARY						
Groups	*Count*	*Sum*	*Average*	*Variance*		
0.1	3	0	0	0		
0.2	3	0	0	0		
0.3	3	0	0	0		
0.4	3	0	0	0		
0.5	3	0	0	0		
ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0	4	0	65535	NA	3.478
Within Groups	0	10	0			
Total	0	14				

Appendix E

E.1 Comprehensive List of Literary Contributions of the Thesis

This appendix lists all the literary contributions which resulted as part of the research conducted in this thesis.

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios. ***Interactive Learning of Alert Signatures in High Performance Cluster System Logs***. In Proceedings of the 2012 IFIP/IEEE Network Operations and Management Symposium (NOMS). Maui, HI, USA. April 2012.

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios, M. Latzel. ***Spatio-Temporal Decomposition, Clustering and Identification for Alert Detection in System Logs***. In Proceedings of the 2012 ACM Symposium on Applied Computing (SAC). Trento, Italy. March 2012.

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios. ***A Lightweight Algorithm for Message Type Extraction in System Application Logs***. IEEE Transactions on Knowledge and Data Engineering (TKDE). Preprint 2011.

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios. ***System State Discovery Via Information Content Clustering of System Logs***. In Proceedings of the 6th International Conference on Availability, Reliability and Security (ARES). Vienna, Austria. August 2011. pp 301 - 306 (Short Paper)

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios. ***A Next Generation Entropy-based Framework for Alert Detection in System Logs***. In Proceedings of the 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM). Dublin, Ireland. May 2011. pp 626 - 629. (Short Paper)

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios. ***Storage and Retrieval of System Log Events using a Structured Schema based on Message Type Transformation***. In Proceedings of the 2011 ACM Symposium on Applied Computing (SAC). Taichung, Taiwan. March 2011. pp 528 - 533.

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios. ***An Evaluation of Entropy Based Approaches to Alert Detection in High Performance Cluster Logs***. In Proceedings of the 7th International Conference on Quantitative Evaluation of SysTems (QEST). Williamsburg, Virginia, USA. September, 2010. pp 69 -78.

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios. ***Fast Entropy Based Alert Detection in Super Computer Logs***. In Proceedings of the DSN Workshop on Proactive Failure Avoidance, Recovery and Maintenance (PFARM). Chicago, IL, USA. June 2010. pp 52 - 58.

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios. ***Extracting Message Types from BlueGene/L's Logs***. Proceedings of the ACM SIGOPS SOSP Workshop on the Analysis of System Logs (WASL). Big Sky, Montana, USA. October 2009.

- *A*. Makanju, A. Nur Zincir-Heywood, E.E. Milios. ***Clustering Event Logs using Iterative Partitioning***. Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). Paris, France. June 2009. pp 1255 - 1264.

- *A*. Makanju, S. Brooks, A. Nur Zincir-Heywood, E.E. Milios. ***LogView: Visualizing Event Log Clusters***. Proceedings of the Sixth Annual Conference on Privacy, Security and Trust (PST). Fredericton, NB, Canada. October 2008. pp 99 - 108.