User Application Monitoring through Assessment of Abnormal Behaviours Recorded in RAS Logs

Byung H. Park, Thomas J. Naughton, Al Geist, CSMD,
Oak Ridge National Laboratory
Raghul Gunasekaran, David Dillow, Galen Shipman,
NCCS, Oak Ridge National Laboratory

ABSTRACT: *Abnormal status of an application is typically detected by "hard" evidence, e.g., out of memory, segmentation fault. However, such information only provides clues for the notification of abnormal termination of the application; lost are any implications as to the application's termination with respect to the particular context of the platform. Restated, the generic exception the application reports is devoid of the overall system context that is captured elsewhere in the system, e.g., RAS logs. In this paper we present an "activity entropy" based application monitoring framework that extracts both facts (events) and context with regard to applications from RAS logs, and maps them into entropy scores that represent degrees of "unusualness" for applications. The paper describes our results from applying the framework to the Cray "Jaguar" system at Oak Ridge National Laboratory, and discusses how it identified applications running abnormally and implications based on the type of abnormality.*

KEYWORDS: application monitoring, log data analysis

1. Introduction

The event logs for a system capture many important details about the platform. However, fuelled by the ever-growing scale of high-end computer systems, the volume and complexity of event logs has reached the point where the manual analysis by a human operator is no longer feasible. This challenge will continue to grow as large-scale systems increase in size. In addition, the highly irregular nature of event logs adds another dimension of difficulty in extracting meanings hidden in the data. For this reason, although equipped with meticulously designed logging systems, mining necessary information from supercomputer event logs is often conducted in rather primitive ways. In its best usage, a log entry is compared against existing databases of message patterns for faults or profiles associated with anomalies learned through extensive effort and accumulated over time. Thus, regular expression matching is the *de facto* tool to extract only the known facts from data that are generated by a sophisticated logging system. For this reason, although embedding rich sources of information regarding system health, resource availability, and behaviour of software at various levels (including user applications), event log data of supercomputers are strikingly underutilized. Except for some primitive fault monitoring efforts, most logs are simply archived and never reopened for analysis thereby leaving invaluable information hidden in the data.

Obtaining information regarding an abnormal application behaviour or failure is not a trivial task. Even hard evidences such as *segmentation violations (segfault)* or *out of memory* are not readily available to users except through a debugging process. Unlike a personal computer, a supercomputer is often regarded as an unpredictable environment for an end user. Since the resources of a supercomputer such as the high speed interconnect network and the file system are shared by many applications (see Figure 1), depending on the allocation or characteristics of applications running concurrently, end users may experience unusual behaviour or an embarrassingly degraded performance for their

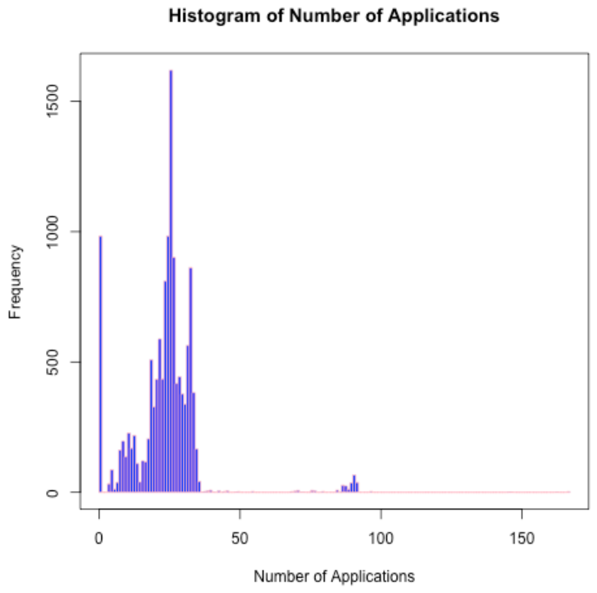

Figure 1 Histogram of number of applications observed within one-minute intervals. Data is obtained from one Jaguar uptime session that spans from Dec. 31, 2010 to Jan.2, 2011.

application. However, very few end users follow detailed traces of their applications to identify clues for suspicious behaviours. In most cases, no details regarding such suspicious behaviours are presented to users.

Processed and viewed through a specific context, log events have been proven to disclose a clear picture of the system. For example, most Lustre log messages report failed I/O attempts between object storage servers (OSS) and object storage clients (OSC). Likewise Basic End to End Reliability (BEER) log messages from the Portals layer include failed communication attempts between compute nodes. If unusually large amounts of these messages are concentrated on very few nodes, it is almost certain that those nodes are in an abnormal state. Lustre messages viewed closely from pair wise contexts may disclose problematic object storage targets. If data is further examined by additional context, more descriptive clues can be drawn. For example, if the set of nodes addressed by BEER messages are allocated to a single application, the fault is likely caused by the application rather than other components in the system. User applications that are not properly tuned for the intended scale tend to impose unforeseeable malignant impacts. This typically involves excessive communication patterns between the nodes occupied by the application or ill coordinated checkpoint attempts. Such an abnormality is difficult to detect unless log event data is analyzed by a context-driven analysis.

This paper introduces a real time application monitoring framework that detects unusual behavior of applications in terms of event activities recorded in logs. For this end, the framework monitors *entropies* of various events not only with respect to compute nodes but also with respect to user applications. More specifically, entropies are measured by mapping each compute node where the event occurs into the application occupying the node. Roughly speaking, an entropy value denotes how evenly the occurrences of the events observed within the current time window are scattered across all user applications. For continuous monitoring, entropies are measured by aggregating all events that occurred within a time window, i.e. a time interval of fixed length that spans from the present time to a certain time in the past.

2. Background

RAS logs, especially those generated through *printk()* are in free form text. Many attempts have been made to capture semantics from these logs by defining regular expressions for the anticipated events. Often these regular expressions are the results of laborious effort by experts, and thus considered to be highly credible to detect mere occurrences of critical events. Systems such as Nagios [1] and SEC [2,3] are real time monitoring utilities based on such hard descriptions of faulty events. These tools are particularly useful for immediate discovery of faults.

Others have used entropy in conjunction with anomaly detection and log data analysis. In [4], the log analysis toolkit, Kerf, was extended to support a Jensen-Shannon algorithm to aid in the ``browsing for anomalies'' in network traffic and security logs. They employ entropy as a means to identify the features (ranking) in their anomaly analysis and visualizer utility. They use these techniques to search and browse data from network log and intrusion detection systems. They use information theory (entropy) to aid in the visualization of network interactions, e.g., source IP address and destination port, etc.

In [5], event logs of an IBM BlueGene/L dataset, which are available from the USENIX Computer Failure Data Repository, were analysed using an algorithm that was enhanced to use entropy. Their tool, Nodeinfo, uses entropy to help automatically identify "significant" events in the log(s). The authors state that the Nodeinfo algorithm is used at Sandia National Laboratory. In [6], Makanzu *et al.* further discuss the cost, accuracy and false positives rate for the entropy enhanced algorithm.

```
[2010-10-02 10:04:33][c16-7c0s5n2]beer: cpu_id 5: nid 11178, cpu 11 has been unresponsive for 240 seconds
[2010-10-02 10:04:33][c16-7c0s4n2]beer: cpu_id 9: nid 11178, cpu 3 has been unresponsive for 240 seconds
[2010-10-02 10:04:33][c16-7c0s4n2]beer: cpu_id 10: nid 11178, cpu 4 has been unresponsive for 240 seconds
[2010-10-02 10:04:33][c16-7c0s4n2]beer: cpu_id 8: nid 11178, cpu 2 has been unresponsive for 240 seconds
[2010-10-02 10:04:33][c16-7c0s5n2]beer: cpu_id 1: nid 11178, cpu 7 has been unresponsive for 240 seconds
[2010-10-02 10:04:34][c16-7c0s3n2]beer: cpu_id 0: nid 11178, cpu 11 has been unresponsive for 240 seconds
[2010-10-02 10:04:35][c16-7c0s5n2]beer: cpu_id 0: nid 11178, cpu 6 has been unresponsive for 240 seconds
[2010-10-02 10:04:35][c16-7c0s4n2]beer: cpu_id 6: nid 11178, cpu 0 has been unresponsive for 240 seconds
[2010-10-02 10:04:35][c16-7c0s4n2]beer: cpu_id 7: nid 11178, cpu 1 has been unresponsive for 240 seconds
[2010-10-02 10:04:35][c16-7c0s5n2]beer: cpu_id 4: nid 11178, cpu 10 has been unresponsive for 240 seconds
[2010-10-02 10:04:35][c16-7c0s5n2]beer: cpu_id 3: nid 11178, cpu 9 has been unresponsive for 240 seconds
[2010-10-02 10:04:35][c16-7c0s4n2]beer: cpu_id 11: nid 11178, cpu 5 has been unresponsive for 240 seconds
[2010-10-02 10:04:35][c16-7c0s5n2]beer: cpu_id 2: nid 11178, cpu 8 has been unresponsive for 240 seconds
                                        ……………
[2010-10-02 20:19:39][c21-7c0s4n3]beer: cpu_id 9: nid 5656, cpu 0 has been unresponsive for 240 seconds
[2010-10-02 20:19:39][c24-0c2s5n3]beer: cpu_id 0: nid 825, cpu 9 has been unresponsive for 240 seconds
[2010-10-02 20:19:39][c24-0c2s5n3]beer: cpu_id 0: nid 2033, cpu 9 has been unresponsive for 240 seconds
[2010-10-02 20:19:39][c24-0c2s5n3]beer: cpu_id 0: nid 5765, cpu 9 has been unresponsive for 240 seconds
[2010-10-02 20:19:39][c24-0c2s5n3]beer: cpu_id 0: nid 5839, cpu 9 has been unresponsive for 240 seconds
[2010-10-02 20:19:39][c24-0c2s5n3]beer: cpu_id 0: nid 12571, cpu 9 has been unresponsive for 240 seconds
[2010-10-02 20:19:39][c16-2c1s3n2]beer: cpu_id 3: nid 10778, cpu 11 has been unresponsive for 240 seconds
```

Listing 1. Two example segments of BEER (basic end to end reliability) log messages.

Their work identifies alerts or entries of interest in the logs from large HPC systems.

Entropy has also been used in other aspects of data mining and search. Mei and Church [7] were primarily concerned with web page mining and web searching. They used entropy to determine the degree of encoding necessary for anticipating successive URLs in web searches. This work explored Microsoft's Live.com data regarding web queries and user access traffic.

3. Technical Details

3.1 Log Entries (Examples)

Listing 1 shows two example segments of BEER (Basic End to End Reliability) messages taken from the *console* log of Jaguar at NCCS. The log spans from October 2, 2010. Two excerpts, separated by roughly 10 hours, illustrate different exemplary cases. The first excerpt shows 13 CPUs from three compute nodes reporting failed communication trials with the same compute node (nid 11178). The second excerpt shows failed communication attempts from three router nodes (to an external file system) to seven compute nodes.

While the second case illustrates a system wide faulty situation when router nodes cannot communicate with compute nodes, the first case clearly shows unusual status of a single node (nid 11178). A further investigation revealed that the same application was running on all of the reporting and reported nodes. This particular case clearly shows an example of abnormal communication behaviour of a single application. Figure 2 includes graph representations of Listing 1, where the original representations (left column, Figure 2-(a),(c)) and their interpreted representations (right columns, Figure 2-(b), (d)) are presented.

3.2 Entropy

In information theory, *entropy* denotes the amount of information that is lost during transmission and often referred to as Shannon entropy [8]. Shannon entropy was originally proposed to measure the amount of information in a transmitted message and thus also understood as the number of bits required to encode the given information. The definition of information is a somewhat vague term and it is in fact the most important step to assign a quantitative meaning to it.

In this paper, we use the notion of entropy to represent the degree of "unusualness." Specifically, for a set $A=\{a_1,a_2,a_3,\ldots,a_n\}$, we define the degree of its "unusual" state with respect to a certain event e to be amount of information required to denote the distribution of e over A. In particular, if an unexpectedly small

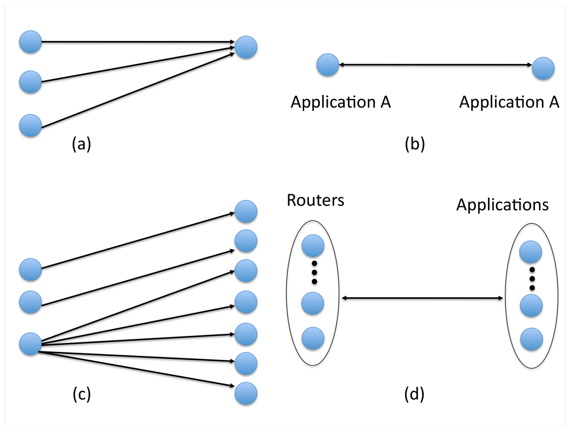

Figure 2 Graph representation of Listing 1. (a) and (c) are graphs of the first and second segments, and (b) and (d) are graphs drawn after mapping nodes into their logical group.

quantity is required to denote the distribution, we decide to call *A* is in an unusual state.

In general information entropy is expressed in terms of a discrete set of probabilities P_i, i.e., the relative fraction of the total occurrences of event *e* that occurs to element $a_i \in A$. Formerly P_i is represented as

$$P_i = \frac{|a_i|}{\sum_j |a_j|} \qquad (1)$$

where $|a_i|$ denotes the number of occurrences of event *e* on element a_i.

Then information entropy is defined as

$$E = -\sum_i P_i \log_2 P_i \qquad (2)$$

where log_2 is the logarithmic function of base two.

Technically speaking, information entropy represents the expected number of bits required to encode the distribution of the occurrences of the event over set *A*. Intuitively, entropy is at its maximum if the occurrences of the event are equally distributed over all elements of *A*, and at its minimum if concentrated on a single element. We therefore define a distribution to be usual or unusual if entropy is high or low, respectively.

3.3 Entropy measured with contexts

Using equation (2) defined in the previous section, system status can be monitored measuring entropies for various events. A typical approach involves computing probability over each compute node, i.e., $a_i \in A$ in equation (1) denotes the *i-th* compute node. In such cases, entropy will indicate whether occurrences of a certain event type occur evenly to a large number of compute nodes or mainly to a small number of compute nodes.

Depending on a context applied, a different view of the system can be captured. We note that entropy measured only over compute nodes may imply a misleading notion under some circumstances. For example, although seemingly scattered across a large number of nodes, if counted from the applications' view, outburst of the event may be mostly due to very few applications. To compute entropy over application, each element a_i in needs to be mapped into the application occupying the node. Formally, equation (1) is slightly modified as,

$$P_k = \frac{\sum_{a_i \in App_k} |a_i|}{\sum_j |a_j|} \qquad (3)$$

where App_k denotes the *k-th* application running in the system.

As aforementioned, certain types of events inherently embed pair wise relations. With such mutual relational context being considered, events can disclose cases when the source of the problem is not the reporting node but the node that is mentioned in the event. Likewise there are cases when we need to capture applications running on the reported nodes.

In summary, applying equation (1) and (2) to capture entropy not over the reporting nodes but over the reported node will also identify whether some few nodes are responsible for the event or not. Likewise mapping the reported nodes into the occupying application and computing entropy using (3) and (2) may reveal the application accountable for the situation.

We therefore propose to monitor the following four types of entropies for an event type.

The distribution of event occurrences over the:

1. *reporting compute nodes* (NodeSRC)
2. *reported compute nodes* (NodeDST)
3. *reporting applications* (AppSRC)
4. *reported applications* (AppDST)

3.4 Continuous measure of entropy values

Since related events can be logged with some latency in between, to compute correct and informative entropy at a given moment may produce a misleading outcome. Instead we measure entropies based on accumulated events within a certain window of time. Thus all event counts in equation (1) represents aggregated counts observed within the window. Formally at time *t*, new occurrences of the event are added and all occurrences added at time *t-w* are subtracted, where *w* denotes the size of the window.

4. Empirical Study and Discussion

This section introduces several exemplary cases that were detected by the proposed framework. All examples were produced from two types of logs, *console* and *netwatch,* from the Jaguar XT5 at the National Center for Computational Science (NCCS), Oak Ridge National Laboratory (ORNL). For the sake of brevity, we will use NodeSRC, NodeDST, AppSRC, and AppDST to denote the four types of entropy as defined in section 3.3.

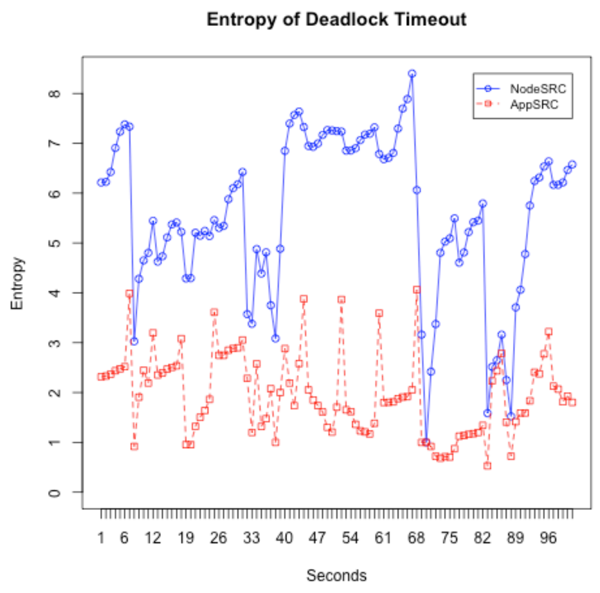

Figure 3 Entropy values of Deadlock timeout measured over 100 second interval (normal case)

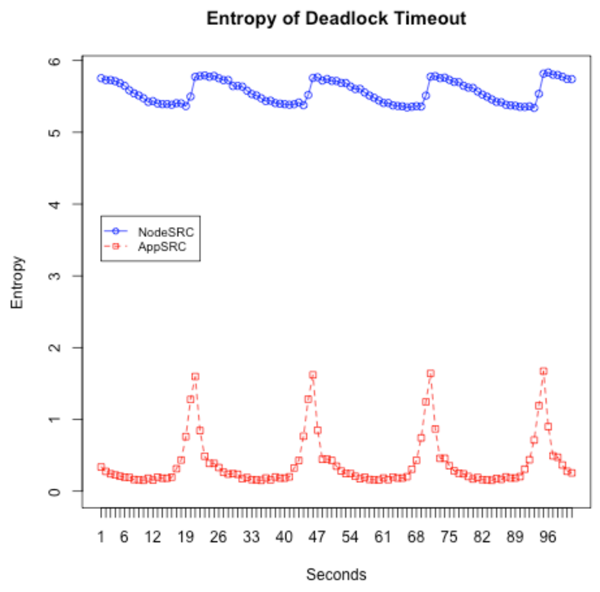

Figure 4 Entropy values of Deadlock timeout measured over 100 second interval (unusual case)

4.1 Deadlock Timeout Events

Deadlock timeout messages logged in a *netwatch* log implicitly denote routing problems between SeaStar chips. For this event type, we consider two types of entropy, NodeSRC and AppSRC, i.e. how evenly the events are distributed over reporting nodes and applications running on them. Figure 3 and 4 are entropy plots that span a continuous 100-second interval each, but illustrating different cases. First Figure 3 shows a typical pattern of an entropy plot during a normal period. Both entropy measures are high throughout the period (mostly greater than one) indicating a system wide problem, i.e. a large number of nodes and applications are affected. On the other hand, Figure 4 shows another set of entropy plot that illustrates an unusual period. As in Figure 3, NodeSRC entropy is constantly high indicating a large number of nodes are involved. However, except for several spots, AppSRC stays close to zero most of the time indicating very few (possibly one) applications are affected by the event storm. Furthermore a periodic pattern of the graph suggests that I/O attempts of the same application have been hampered due to the same cause.

4.2 Lustre Events

Lustre messages are largely about failed I/O attempts between Object Storage Clients (OSCs) and Object Storage Targets (OSTs) through Object Storage Servers (OSSes). A close examination of the four entropy plots reveals a clear picture regarding the system status. For

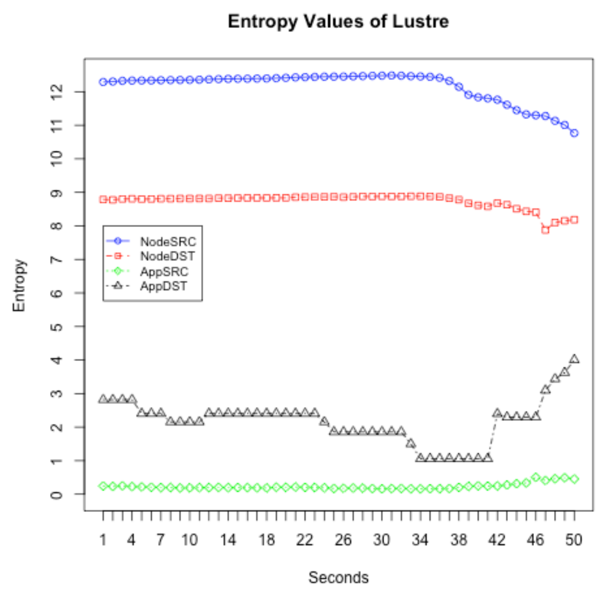

Figure 5 Entropy plots of Lustre events during a period when communications between all OSSes and all applications are blocked.

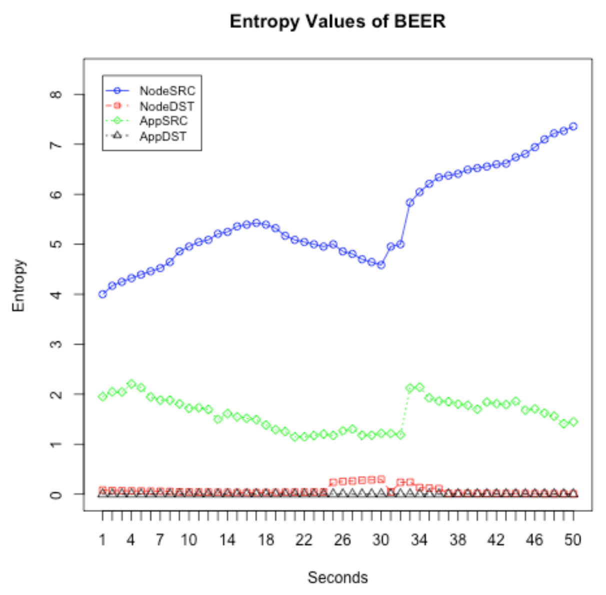

Figure 7 Entropy plots of BEER events during a period when a single router stops responding.

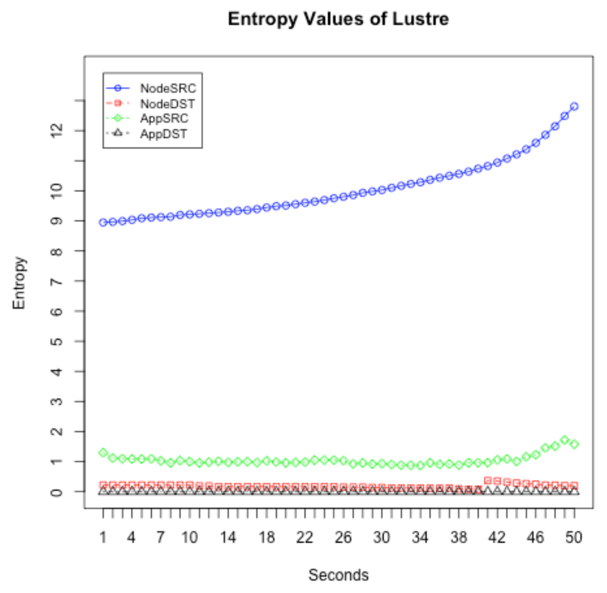

Figure 6 Entropy plots of Lustre events during a period when communications between applications and an OST is blocked.

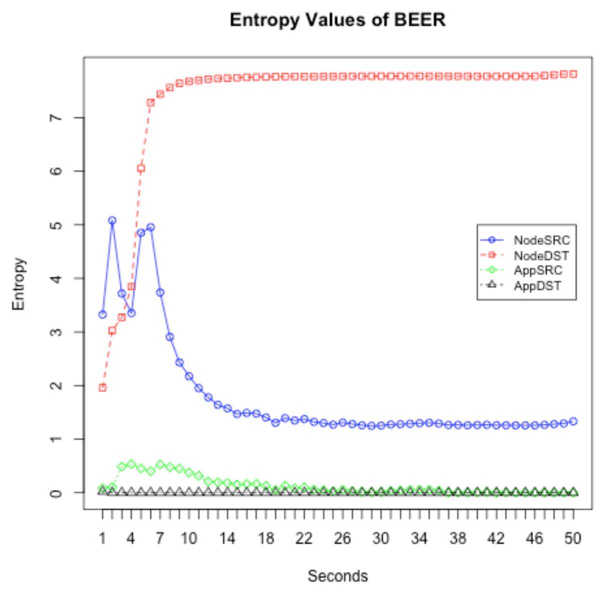

Figure 8 Entropy plots of BEER events during a period when communication attempts between the same application fail.

example, large values of both NodeSRC and NodeDST indicate system wide communication problems between OSCs (or nodes) and OSTs. Figure 5 shows a 50 second interval of such a case. Low AppSRC but high AppDST values in Figure 5 denote that very few applications are involved in generating the events but a large number of applications are associated with the nodes that are reported. A thorough investigation disclosed that the sources of the events are OSSes. In other words, all OSSes report failed communications with virtually every application in the system.

When very few OSTs are the source of I/O problems, both NodeDST and AppDST are measured low. Additionally, AppSRC indicates whether a small or large number of applications are affected by the problem OSTs. Figure 6 shows entropy plots measured over a 50 second interval when a single OST is the problem source of the failed I/O attempts. As shown, AppSRC is around 1 throughout the interval, which suggests that multiple applications suffer from one problematic OST.

4.3 BEER Events

Basic End to End Reliability (BEER) errors are reported when communication attempts at the Portals layer face difficulties. As introduced in Listing 1, a BEER message reports a failed communication between two nodes. Therefore a close inspection on the four entropy measures will disclose the status of the system when communication failures of non-negligible scale seem to be prevalent. For example, a small NodeDST value indicates that most failed communication attempts have the same destination. Depending on the AppSRC value, a more detailed picture can be obtained. A large AppSRC value when NodeDST is small indicates that multiple applications suffer from the same cause, i.e. the problem node is common. This scenario is shown in Figure 7 where multiple nodes and applications are experiencing failed communication with the same router node.

Inspection of the four entropy measures of BEER events is particularly important in sifting applications that involve unusually intense communication activities. In such a case, AppSRC and AppDST are expected to be close to zero while NodeSRC and NodeDST are high. Figure 8 illustrates one such case.

Conclusion

RAS logs embed clues for not only system wide anomaly but also unexpected behaviours of user applications. However, due to the unstructured nature and extraordinary volume of logs, detecting an abnormal status of the system or applications and mining its plausible cause has been severely restricted. For this reason, in particular, virtually no information is provided to end users regarding unusual behaviours of their applications.

In this paper we introduced a real-time system-monitoring framework that highlights both the unusual status of the system and that of user applications. The framework, which is based on measuring Shannon's entropies in different contexts, has been applied to capture salient signatures of event distributions over compute nodes and user applications. Although preliminary, the results of the initial analysis on the RAS logs from Jaguar at NCCS illustrate that the framework is able to detect suspicious periods and simultaneously suggest possible reasons.

The current prototype is still in a preliminary stage and needs to be further refined in several directions. In particular, incorporation of not only predefined "hard" event types but also inferred event types is highly desired. We are currently investigating various log data clustering algorithms for this end.

Acknowledgments

The authors would like to thank Don Maxwell (NCCS) and Jeff Becklehimer (Cray Inc.) for their help. ORNL's work was supported by the U.S. Department of Energy, under Contract DE-AC05-00OR22725.

References

[1] Wolfgan Barth, "Nagios: System and Network Monitoring", No Starch Press; 2 edition, 2008

[2] Risto Vaarandi. "SEC - a lightweight event correlation tool", In Proceedings of the IEEE Workshop on IP Operations and Management, pages 111–115, October 2002.

[3] Jeff Becklehimer, Cathy Willis, Josh Lothian, Don Maxwell, and David Vasil. Real Time Health Monitoring of the Cray XT Series Using the Simple Event Correlator (SEC). In 2007 Cray User Group Conference, May 2007. Seattle, Washington, USA.

[4] Javed Aslam and Sergey Bratus, "Semi-supervised Data Organization for Interactive Anomaly Analysis", In Proceedings of the Fifth International Conference on Machine Learning and Applications (ICMLA) , December 2006.

[5] Adetokunbo A.O. Makanju and A. Nur Zincir-Heywood, and Evangelos E. Milios, "Fast Entropy Based Alert Detection in Supercomputer Logs", In Proceedings of the 2nd DSN Workshop on Proactive Failure Avoidance, Recovery and Maintenance (PFARM), IEEE, 2010.

[6] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos 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), IEEE, 2010.

[7] Qiaozhu Mei and Kenneth Church "Entropy of search logs: how hard is search? with personalization? with backoff?", In Proceedings of the International Conference on Web search and Web data mining (WSDM), ACM, 2008.

[8] C.E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, pp. 379–423, 623-656, July, October, 1948