Correlating Log Messages for System Diagnostics

Raghul Gunasekaran, David A. Dillow, Galen M. Shipman
Don Maxwell, Jason J. Hill
Oak Ridge Leadership Computing Facility
Oak Ridge National Laboratory
{gunasekaranr,dillowda,gshipman,maxwellde,hilljj}@ornl.gov

Byung H. Park, Al Geist
Computer Science Research Group
Oak Ridge National Laboratory
{parkph,gst}@ornl.gov

Abstract

In large-scale computing systems, the sheer volume of log data generated presents daunting challenges for debugging and monitoring of these systems. The Oak Ridge Leadership Computing Facility's premier simulation platform, the Cray XT5 known as Jaguar, can generate a few hundred thousand log entries in less than a minute for many system level events. Determining the root cause of such system events requires analyzing and interpretation of a large number of log messages. Most often, the log messages are best understood when they are interpreted collectively rather than individually. In this paper, we present our approach to interpreting log messages by identifying their commonalities and grouping them into clusters. Given a set of log messages within a time interval, we group the messages based on source, target, and/or error type, and correlate the messages with hardware and application information. We monitor the Lustre log messages in the XT5 console log and show that such grouping of log messages assists in detecting the source of system events. By intelligent grouping and correlation of events in the log, we are able to provide system administrators with meaningful information in a concise format for root cause analysis.

1 Introduction

Understanding system failures and isolating problems from log data in large-scale computing systems is a complex and tedious task. System logs often have large amounts of redundant information which inhibits interpretation of these messages. Further complicating matters, these log messages are often highly unstructured.

Though the volume of log generated is large, the messages by themselves often do not provide sufficient information for identifying the root cause of a problem [7]. To gain a complete understanding of a specific event and identify the root cause, the system administrator needs an extensive knowledge of the system and its current state. Making matters worse, the sheer volume of system log data often necessitates reducing the fidelity of system logging. At the current scales of operation for the Oak Ridge Leadership Computing Facility (OLCF), it is impractical to run many of our systems at normal logging levels in order to collect salient details about system events as the data volume is simply too great. To reduce the monitoring load of our system data analytics platform, we intially focus on the XT5 console log and the Lustre file system [1] messages in particular. These console log messages are generated via printk statements within the Linux kernel. These messages are unstructured and vary over time as the format of the message may change for each kernel or system software release, often becoming increasingly complex. Understanding the structure of these error messages is essential to interpreting them, which requires extensive pre-procesing to render the data into a form usable for clustering. While our approach of clustering log messages is generic, interpreting these log messages in context requires knowledge of the OLCF infrastructure [4].

In this paper, we define a three step process of in-

terpreting the Lustre file system error messages from the console log. The objective is to present log information in a concise format without loss of information, enabling easy interpretation and isolation of problems. First, we preprocess the log messages, parsing the Lustre error messages using knowledge of the structure of the error messages gained from the source code. Second, we cluster the log messages within a time widow identifying a common feature set. This provides us with a precise summary of events in the log within a given time window. Finally, these clustered log messages are run through a time-series analysis to capture system-wide event patterns during a period of system uptime, as well as associated with job information to find trends tied to individual applications. This paper details the above methods and presents our analysis of the Jaguar XT5 logs from the months of December 2009 to February 2010.

Related Work: Conventionally, log messages have been associated with temporal data mining techniques. These methods are best suited for transaction logs where the frequency and sequence of events are of interest. In HPC logs, a burst of log messages within a short time period would define a failure and these sets of log messages repeat at regular intervals until the problem is fixed. The first burst of information is important for isolating the problem, while the rest of the messages are often redundant. To further analyze the problem, it is important to understand the log messages preceeding the reported event.

Recent studies have focused on machine learning and statistical methods of analyzing and detecting system failures. SLCT (Simple Logfile Clustering tool [8]) is a data clustering paradigm for mining event patterns in log data. An *apriori* algorithm, the first stage is to generate a count of all unique words in the log, identify log messages containing words above a threshold value, and then clustering those log lines. This method is based on the assumption that events of interests occur in bursts and this method ignores errors with low frequency. In the machine learning paradigm described in [9], the log message structures are parsed from the source code and a feature vector is constructed as a sequence of log messages. Using principal component analysis techniques, deviations of the run-time log from the predefined vectors are identified – any variance is defined as an anomaly. This approach is based on the assumption that the system supports logging of all events, which may be impractical for large systems like Jaguar.

Nodeinfo, an entropy based anomaly detection system, was proposed in [5]. In *Nodeinfo*, the log messages are classified as an *alert* or not, and are tagged to quantify the importance. Then, the entropy of every node in the system is quantified based on the number of occurrences of alerts within a given time period. It is presumed that all nodes operate similarly, and the entropy of each node should be uniform. Any variance of entropy would be categorized as an alert/anomaly. This technique assists in identifying nodes causing an issue, but does not achieve high efficiency in interpreting the log messages. Similarly, the models proposed in [6] and [2] group log messages and use predictive techniques under the assumption that the logs carry all event information and occur in bursts.

2 Log Pre-processing

In our initial work we focus on the Lustre file system error messages in the console log. Figure 1 shows the steps involved in extracting information from the log for further analysis. We parse CDEBUG messages from the Lustre 1.6 source code categorized as *D_EMERG* and *D_ERROR*. We then define regular expressions for each individual message of interest, identifying the various components of information. The log parser, written in python, parses console log messages based on matching regular expressions and also uses OLCF system specific information to derive more attributes for the data parsed from the log. The site specific information includes mapping node ID to its NID, IP address and node type, IP address of InfiniBand routers, and mapping file system storage server IP to it's type (oss,mds or mgs). The parsed log is stored in a MySQL database for further analysis.

We explain the process with the help of two examples as shown in Table 1. The first error message is a node reporting that it is unable to establish communication with a specific router, and the second error message is from a node that is unable to do a write operation on an OST. The attributes as shown in the table are extracted with prior knowledge of the structure of the message defined as a regular expression. Every log entry starts with a *timestamp*, time the log was generated, and *sourceID*, the node that generated the message, followed by the actual error message. From the *sourceID*, we identify the NID number (*sourceNID*) and node type (*sourcetype*) information. In general, the source node can be a compute (cnode), lnet router (rtr), login, batch or service (svc) node. The NID number, which is of integer datatype, is useful while analyzing logs compared to *sourceID* field which is of character datatype. Also, most log messages carry process ID, and details of the

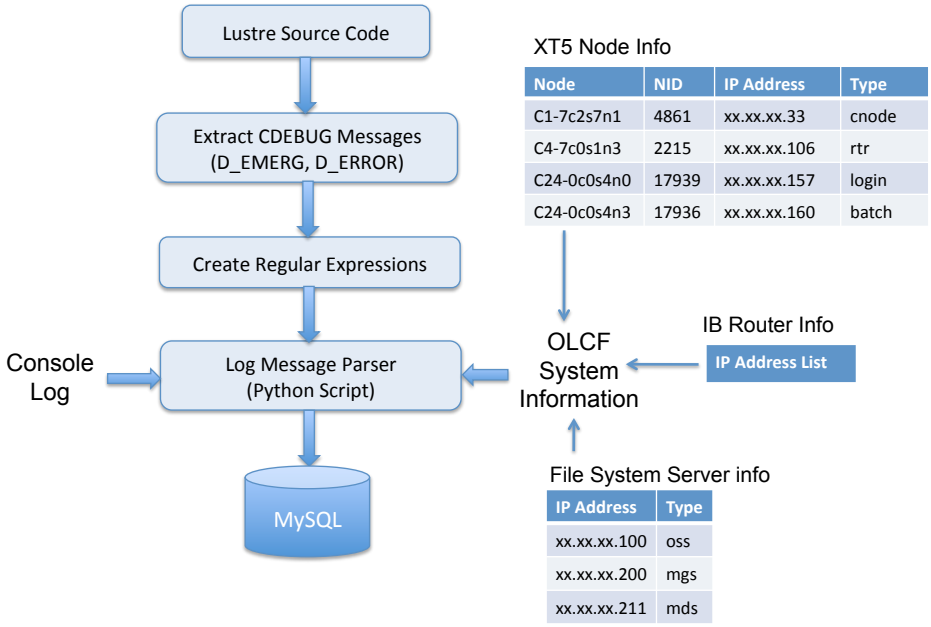

Figure 1: Pre-processing Log Messages

Error Message [2010-01-13 07:22:05][c16-3c1s4n0]LustreError: 16149:0:(ptllnd_peer.c:903:kptllnd_peer_check_bucket()) Could not send to 12345-18235@ptl1 after 250s (sent 293s ago); check Portals for possible issues

Processed Error Message
timestamp 2011-01-13 07:22:05
sourceID c16-3c1s4n0 *sourceNID* 10832 *sourcetype* cnode
processID 16149 *module* lnet
modulefile ptllnd_peer.c *modulefunction* kptllnd_peer_check_bucket()
target1 18235 *target1type* rtr
errormsg check Portals for possible issues

Error Message [2010-01-24 12:04:02][c10-4c1s0n1]LustreError: 25704:0:(events.c:55:request_out_callback()) @@@ type 4, status -5 req@ffff8101f428b800 x1872401/t0 o4->widow1-OST0008_UUID@xx.xx.xx.105@o2ib:6/4 lens 386/480 e 0 to 1 dl 1264353000 ref 3 fl Rpc:/0/0 rc 0/0

Processed Error Message
timestamp 2010-01-24 12:04:02
sourceID c10-4c1s0n1 *sourceNID* 8609 *sourcetype* cnode
processID 25704 *module* ptl
modulefile events.c *modulefunction* request_out_callback()
target1 widow1-OST0008 *target1type* ost
target2 xx.xx.xx.105 *target2type* oss
errormsg 4 failed
errorcode 5

Table 1: Example Log messages and their parsed contents

source file and function that generated the error message. In Table 1, the attributes *processID*, *modulefile* and *modulefunction* identify the process, file and function in Lustre that generated this error message. From the *modulefile* information we can identify which Lustre module generated the error. In general, the module can be Lustre networking (lnet), Lustre's Portals driver (ptl), object storage (obd), locking (ldlm), striping (lov) or general client (llite).

Then we identify the entity the source is complaining on, which is the target, and the part of the log that best describes the error. For some error messages there can be more than one target, captured as *target1* and *target2*. The first error message points to only one target, *target1* is identified by NID 15235, and its type information is captured as *target1type*. In the second error message there are two targets, *target1* is an OST and *target2* is an OSS. This parsing helps analysis, as we can isolate if the OSTs mentioned in a log cluster belong to a single OSS [1]. The specific error is recorded as *errormsg*, for the first error message it is 'Check Portals for possible issues', and for the second error message we parse for the number suffixed to 'o', which translates to operation 4 – OST_WRITE, an object write operation. A few log messages specify the exit status or return code, usually followed by keyword 'status' or 'rc' as in the second example. The exit status is stored as *errorcode* attribute. The remaining parts of the log message are stored in the field *errordesc*, for "error descriptions". Thereby, the complete log message can be restitched from a row in the database.

3 Clustering Event Logs

The goal of clustering the system log data is to extract actionable information from the synthesis of the parsed log messages and knowledge of the system's state and environment. The system state includes details of the applications running on the system and the sets of nodes allocated to these applications. The system environment includes the condition of system components (active/failed) and hardware dependencies such as the mapping between OST, OSS, and RAID controllers. Our clustering approach is performed in two steps; first, messages are grouped within a window of time, and then those groups are combined across time windows.

[1] With additional information about the system configuration, we can correlate errors on multiple OSSes with a common RAID controller to find issues with the back-end storage.

3.1 Clustering within a time window

1. A set of one-to-one mappings based on source and target (target1) types is generated for a given time window. This results is a general overview of which categories of nodes are reporting errors, and the corresponding node types. This process is iterated from a time window of one minute, and is repeated with increasing windows of time (one minute) until the log messages are reduced to one-third of the original number of records. This is based on the observation that each event is most often represented by three consecutive log messages. The time window is limited to the major Lustre timeout setting [2] to help identify periodic messages resulting from lost RPCs.

2. For the time window identified, a summary of the log messages is generated in terms of *source type, target1 type, module* and *errormsg*. This provides a general overview of the log within the time frame.

3. For each source type seen in the window, we then generate a list of distinct source NIDs and error message pairings. We generate a second list for each type of the first target. These lists allow identification of commonalities and aid isolation of the failure point.

4. We then generate a one-to-one mapping between *target1* and *target2* for the window as well as a list of nodes mentioned as *target2*. This provides a list of nodes (usually OSSes) providing the service described in *target1* (usually OSTs). These lists allow a grouping of errors on OSTs sharing a common OSS to be attributed to the OSS itself. This can be extended using knowledge of the mapping from OSS to RAID controllers to identify the probability that the RAID controller is the root cause of failure.

5. Finally, for each compute node that is the source or target of a message, the node is mapped to the application that is executing on it. This allows correlation of events to application runs and an analysis of errors associated with each distinct application.

3.2 Clustering across time windows

The above process generates clusters of log messages identifying commonalities within discrete time

[2] Currently 600 seconds for OLCF systems.

sourcetype	target1type	module	errormsg
cnode	ost	ptl	Connection lost
cnode	mdt	ptl	Connection lost
cnode	ost	ptl	Connection restored to service
cnode	oss	ptl	Request Timed Out
cnode	mdt	ptl	Connection restored to service
cnode	MGS	ptl	Connection restored to service
cnode	MGS	ptl	Connection lost

Table 2: A clustered view of log messages

sourcenid	sourcetype	target1type	module	errormsg
1947	rtr	oss	ptl	Conn race

Table 3: A single entry representing 19 log messages

sourcenid	sourcetype	target1	target1type	module	errormsg
1947	rtr	xx.xx.xx.184	oss	ptl	Conn race
1947	rtr	xx.xx.xx.121	oss	ptl	Conn race
1947	rtr	xx.xx.xx.17	oss	ptl	Conn race
1947	rtr	xx.xx.xx.116	oss	ptl	Conn race
1947	rtr	xx.xx.xx.188	oss	ptl	Conn race
1947	rtr	xx.xx.xx.191	oss	ptl	Conn race
1947	rtr	xx.xx.xx.162	oss	ptl	Conn race
1947	rtr	xx.xx.xx.18	oss	ptl	Conn race
1947	rtr	xx.xx.xx.128	oss	ptl	Conn race

Table 4: A drill down of the single entry from the above table

frames.We then colapse those clusters into larger windows subject to two restrictions: the segments of time must be contiguous, and error content must be the same.

For example, given a burst of identical log messages representing a single event with 5000 messages over five minutes, our method generates five clusters of messages that occur sequentially in continuous time. These clusters are then colapsed into a single window of five minutes duration. This is similar to analysis using a sliding time window, and retains information about the periodicity of the messages.

Given this sequence of temporally discontiguous clusters of messages, we can then check for periodic application or hardware errors. Hardware errors are detected by looking for similar errors within the period of system uptime containing the cluster under examination. Behaviour analysis is performed against the last ten executions of the application associated with the current messages, searching for similar patterns of error messages.

4 Results

The following section details the various inferences we draw from log messages by pre-processing and clustering log messages.

4.1 Correlating logs

Even though we support a small subset of debug messages in our prototype system, the volume of log messages generated is overwhelming. This deluge of data is mitigated by the observation that the bulk of the messages reduces down to a few lines of useful information. In our experience with Lustre, a burst of messages generally is a consequence of an event that occurred in the relatively distant past – a time scale of a few minutes. For example, Table 2 shows a summary of error messages occurring approximately 30 minutes after system boot. The 7 clusters depicted were reduced from over 3000 raw log messages. A set of compute nodes lost their connection to the file system servers and re-established their connections. At this point in time, no application

sourcetype	target1type	errormsg
cnode	oss	Request Timed Out
cnode	ost	400 failed
cnode	ost	Connection lost
cnode	ost	Connection restored to service

Table 5: Log entries when compute nodes choose a faulty path

sourcetype	target1	target1type	errormsg
batch	c14-0c0s6n0	rtr	PTL_NAL_FAILED(4)
svc	c14-0c0s6n0	rtr	PTL_NAL_FAILED(4)
login	c14-0c0s6n0	rtr	PTL_NAL_FAILED(4)
cnode	c14-0c0s6n0	rtr	PTL_NAL_FAILED(4)

Table 6: The inital burst of messages identifying a common target

sourcetype	target1	target1type	errormsg
cnode	c14-0c0s6n0	rtr	check portals
login	c14-0c0s6n0	rtr	Timing out
batch	c14-0c0s6n0	rtr	Could not get credits for

Table 7: The periodic messages (every 250 seconds) untill the problem is fixed

was running on the machine. To determine the cause for these messages, we look at the prior messages. Ten minutes before these messages, there were 19 Lustre information messages informing us that a specific router had a connection race with a number of storage servers. Our clustering approach captured it as a single entry, Table 3, representing a more detailed view shown in Table 4 indicating transient errors from the Lustre Portals driver between a particular router and a few OSSs. Our time series analysis – using domain knowledge – was able to associate the earlier transient errors with the more recent connection loss messages. In this example, we did not see any further error messages for that particular router during the normal system run. This identification of a transient fault is most useful when multiple fault events are occuring simultaneously and can help the system administrator to separate the important from the unimportant. Future work includes determining the value of information gained from this type of correlation and using that to decide if an alert should be delivered to a system administrator in real-time.

4.2 Hardware Anomalies

Analysis of the Lustre log messages can also help isolate hardware anomalies and identify the location of the fault. During one system run, approximately twenty three thousand log messages occured in the span of three minutes. The messages, summarized in Table 5 and Table 6, indicate that compute nodes lost connection with the object storage servers, and was noticed due to periodic heartbeat messages rather than user traffic – *400 failed* indicates OBD_PING failed. The service (svc), compute, batch and login nodes all complained about a specific router node, with error PTL_NAL_FAILED(4). These were followed by over two million log messages in a three hour period, summarized in Table 7. The first set of log messages indicates that a single router is at fault. The long trail of messages following the initial set is due to the clients trying to reestablish communication with the router every 250 seconds. That specific router experienced a kernel panic approximately 4 minutes after the first set of error messages and needed to be rebooted.

While this particular example may not be especially helpful to an experienced system administrator – the logs would show the kernel panic of the router – it demonstrates the ability of the method to pinpoint the source of the fault quickly in an avalanche of data. This capability is of tremendous value particulary when the root cause of failure has not previously been categorized and actively monitored for. For example, we are able to isolate failures/errors occurring on specific OSS or RAID controllers, in which case the log messages complain about each OST(s) attached to the server(s). While this analysis is currently reactive and does not provide

predictive results, future work integrating historical data to predict failures holds promise.

4.3 Application Patterns

We correlate error messages to applications running on those nodes to identify application behaviors which affect its normal run. Jobs are terminated for a number of reasons: application defects, walltime exhaustion, hardware failure, and system software errors are among the most common. Failures due to application defects such an *out of memory errors* may produce spurious Lustre messages which complicate analysis and are often best removed from the log clustering. However, failures due to system software errors and hardware faults generally result in entries in the console log that are key to analyzing the root cause. In most of these cases, they user would see some indication of the failure in the console output of their job.

In one observed case, a user reserved a large number of compute nodes and launched multiple apruns in parallel on a small subset of the compute nodes. The aprun command defined a wall-time limit of 1800 seconds. Despite the expected running time, more than half of the jobs were terminated within a few seconds by a fatal, uncatchable signal (SIGKILL). The killed were associated with the Portals error PTL_NAL_FAILED and Lustre operation 35 failure, MDS_CLOSE. This occurred for more than two months, with no known system issues that could explain the failure. It appeared that the application had a role in causing or otherwise exposing a fault.

In a second case, we found a specific application had similar Lustre errors for three consecutive weeks, but the errors did not terminate the application run. The application had successful runs prior to this period, and such patterns where observed only within the three weeks. The nodes on which the application was running periodically reported OST_STATFS failures, which are likely a result of the *statfs* system call. The user acknowledged that they were trying new libraries during that time period, which may have caused or otherwise exposed a fault witin the system.

In both cases the user had made no reports of system errors or otherwise indicated that problems during runtime existed. A full root-cause analysis was not possible in either of these cases as both issues appear to have been transient (although over a large duration) and system software upgrades and application library changes have appeared to either correct or mask these erors. Though raising more questions than answers due to the limitations imposed by using months old data, our methods highlighted specific correlations between application behavior and Lustre errors. While contacting users about such old activities often leads to puzzlement due the difficulty recollecting specific changes, moving the analysis close to real-time will permit more immediate investigation of the root causes of the Lustre errors. This will lead to higher quality bug reports to the vendor or improved application code, depending on the actual source of the fault. As our techniques are further refined based on historical datasets we will deploy near real-time analysis methods within our infrastructure enabling timely reporting of issues that may otherwise go unnoticed or otherwise unreported.

5 Conclusion

Deriving meaningful, actionable information from the deluge of log data generated by modern leadership class computing platforms is essential to providing reliable world-class computing to a growing scientific community. Unfortunately, the sheer volume and complexity of system log data inhibit the use of the wealth of data contained in these logs for root cause analysis and response. To this end we have described techniques for system log transformation to structured data and clustering techniques for data analysis. These techniques reduce total system log volume while preserving and enhancing the diagnostic value of this data. While the techniques described in this paper have only been prototyped, our intent is to continue to develop these techniques as the amount of log data to be analyzed will only grow over time. This work provides insight into faults on existing systems, and its effectiveness will increase as we work to process log streams in real-time to enable near real-time response to system failures.

6 Acknowledgements

The authors would like to acknowledge the efforts of the RAVEN [3] project. Their assistance in our initial analysis work helped interpret the results and visualize the impact on the system after failures occured.

References

[1] Lustre Filesystem. `http://wiki.lustre.org`.

[2] A. Makanju, A.N. Zincir-Heywood, E. E. Milios. Clustering event logs using iterative partitioning. In *ACM SIGKDD International conference on Knowledge discovery and data mining*, 2009.

[3] Byung H.Park , Guruprasad Kora, Al Geist, Junseong Heo. RAVEN: RAS data analysis through Visually Enhanced Navigation. In *Cray User Group Conference*, 2010.

[4] Galen M.Shipman, David Dillow, Sarp Oral, Feiyi Wang. The Spider Center Wide File System: From Concept to Reality. In *Cray User Group Conference*, 2009.

[5] J. Oliner, A. Aiken, and J. Stearley . Alert Detection in System Logs. In *Proceedings of the International Conference on Data Mining (ICDM)*, 2008.

[6] Y. Liang, Y. Zhang, H. Xiong, Hui, R. Sahoo. Failure Prediction in IBM BlueGene/L Event Logs. In *Proceedings of the International Conference on Data Mining (ICDM)*, 2007.

[7] Adam Oliner, Jon Stearley. What Supercomputers Say: A Study of Five System Logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2007.

[8] Risto Vaarandi. A data clustering algorithm for mining patterns from event logs. In *IEEE IPOM03 Proceedings*, 2003.

[9] W. Xu, L. Huang, A. Fox, D. Patterson, M. Jordan. Mining Console Logs for Large-Scale System Problem Detection. In *3rd Workshop on Tackling System Problems with Machine Learning Techniques(SysML)*, 2008.