A Diagnostic Utility For Analyzing Periods Of Degraded Job Performance

Joshi Fullop and Robert Sisneros
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Urbana, IL USA
Email: {fullop, sisneros}@illinois.edu

***Abstract*—In this work we present a framework for identifying possible causes for observed differences in job performance from one run to another. Our approach is to contrast periods of time through the profiling of system log messages. On a large scale system there are generally multiple, independently reporting subsystems, each capable of producing mountainous streams of events. The ability to sift through these logs and pinpoint events provides a direct benefit in managing HPC resources. This is particularly obvious when applied to diagnosing, understanding, and preventing system conditions that lead to overall performance degradation. To this end, we have developed a utility with real-time access to the full history of Blue Waters data where event sets from two jobs can be compared side by side. Furthermore, results are normalized and arranged to focus on those events with greatest statistical divergence, thus separating the chaff from the wheat.**

***Keywords*-System Logs; Data Filtering**

I. Introduction

In this work we present a framework for identifying possible causes for observed differences in job performance from one run to another. Our approach is to characterize and contrast periods of time through the profiling of system log messages. On a large scale system there are generally multiple, independently reporting subsystems, each capable of producing mountainous streams of events. Log-worthy events are anything from those associated with catastrophic system failures down to the multitudinous and mundane machine minutiae. The ability to sift through these logs and pinpoint events provides a direct benefit in managing HPC resources. This is particularly obvious when applied to diagnosing, understanding, and preventing system conditions that lead to overall performance degradation. Such conditions are the specific target of this work. In fact, the motivation of the development efforts were undertaken to address the almost constant question from users as to why one of their jobs performed better than another.

To begin this presentation we describe the Blue Waters logging configuration and enumerate the various subsystems whose logs are consumed. These include but are not limited to the mainframe via Crays Lightweight Log Manager (LLM), the Bright Cluster managed External Services Management Servers (ESMS), three Sonexion-based Lustre file systems, a HPSS near line storage system, Moab and Torque scheduling services and various network switching

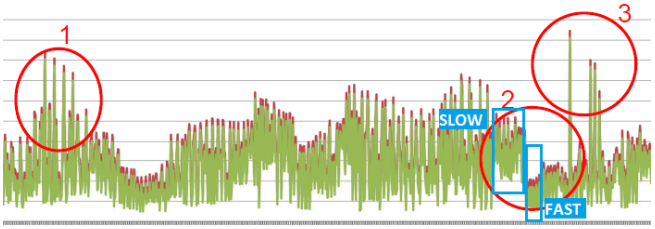

Figure 1: Runtime variations across multiple job runs.

equipment. Peak log volumes we have experienced so far were in excess of 200GB in a single day. We also present the use of our Hierarchical Event Log Organizer (HELO) to classify and ingest events into a database at sustained rates exceeding 20,000 log messages per second. The benefits of using a database to organize this information are numerous, but the one most critical, when combined with integer-based tagging of log messages, is indexed searching. To this end, we have developed a utility with real-time access to the full history of Blue Waters data. Specifically, we compare system-wide events during two jobs, one with normal performance and another with marginalized performance. And we describe the different forms of contrast used in comparing the fingerprints of the periods in question. Furthermore, results are normalized and arranged to focus on those events with greatest statistical divergence, thus separating the chaff from the wheat.

II. Background

The Blue Waters supercomputer generates many multiple gigabytes of log data per day. Sifting through this amount of data from various perspectives with different agendas requires almost as many mining techniques as there are questions posed upon it. Without endeavoring to answer all the possible questions, we will constrain our paper to discussing one of the most commonly posed questions by the user community: “Why did my job run fast during this time and not at another?”. This question can be more generically posed in terms of log assessment as “What events occurred during one time period versus another?”.

One of the overriding general premises is that log messages are generated when something goes wrong, and is silent when things are functioning properly. Or in some

JobID	Start	End	Time (s)	Avg Node load	Job Starts	Node Starts	Job Ends	Node Ends	Max Msg Rate	Avg Msg Rate	Notes
162182	1355593017	1355599817	6800	24237.4035	122	34717	107	34235	200	51	
162321	1355599886	1355600725	839	24786.0769	20	883	24	10729	17786	1,105	ERROR A (see below)
162343	1355606271	1355612557	6286	23899.2095	101	31808	100	32551	19134	106	
162381	1355602365	1355606144	3779	24515.1429	87	21469	95	19614	200	109	
162472	1355606261	1355611625	5364	23930.0778	80	19709	80	20280	19134	114	
162562	1355611866	1355617019	5153	23830.4941	80	42705	83	38445	429	75	
162592	1355617116	1355622019	4903	22191.2317	79	52838	80	53564	438	69	
162659	1355617094	1355622254	5160	22182.5698	87	59068	88	58850	438	68	
162738	1355622137	1355628341	6204	23155.0777	108	40417	103	37343	18233	65	
162745	1355622332	1355628853	6521	23276.4404	114	37621	111	37721	18233	65	
162841	1355628541	1355632353	3812	22772.0476	69	21522	77	23330	200	56	
162854	1355628919	1355634644	5725	23280.4000	104	36985	110	35323	200	55	
162922	1355651132	1355654743	3611	16737.9667	62	21184	63	23181	200	51	
162967	1355651133	1355655061	3928	16524.8923	67	20992	72	24404	200	52	
163331	1355662727	1355666557	3830	25147.1406	62	9427	65	10428	200	54	
163340	1355683811	1355688705	4894	25003.8765	76	17586	75	16239	200	56	
163543	1355666651	1355670806	4155	23483.7826	44	12901	65	17109	200	51	
163614	1355684350	1355691585	7235	25077.5500	107	22732	105	22712	200	52	Ran past wallclock
163898	1355688826	1355696042	7216	24672.1074	121	20037	112	20520	200	52	Ran past wallclock

Figure 2: A comparison of the outcomes resulting from multiple runs of the same job.

cases, the frequency of regularly occurring log messages that indicate minor issues increases. Of course, there are counter cases to both of these, but those are considered far more rare instances. None the less, we provide a method of identifying them as well.

Another difficulty in analyzing log data on a system with the complexity of BlueWaters is defining a normal by which any given period can be compared. This is also very true of metric data, but is a topic for another discussion. Using "standardized" jobs with exact workload on the same number of nodes gives us at least the beginnings of a basis for comparison. Ideally, the same placement pattern would be beneficial, but that is not always practical.

These principals were the ones utilized in a study performed jointly by members of NCSA and Cray in attempt to discover cause and correlation between various measurable factors and job trial performance, as is evident in Figure 1. This consistency analysis was unable to divine a singular factor with a consistently high correlation value across job types and periods of execution. However, that is the nature of extremely complex systems. One theme that persisted was that performance was certainly impacted by what else was occurring on the system at the times in question. Initially, that drove the group to expand its consideration into factors like: the number of other jobs started or ended during each job in question, the number of nodes included in summation of those job starts and ends, the number of other nodes in use during that time period, the number of critical events occurring, etc. Figure 2contains this information for one such set of runs. Eventually, the broader question of how do the log events compare between two observed periods of performance came to the forefront. This was the spark that initiated the following work.

III. System Logs

Before we can describe our methods, we first need to describe the scope and scale of our data and the infrastructure used to ingest it.

A. Infrastructure

By method we separate each source to communicate to NCSA's Integrated Systems Console (ISC) over its own port. Syslog-NG is used to receive, map and route these feeds to the appropriate parsers, handlers and archives. One significant issue is that the various log sources are facilitated by as many versions of remote logging software. These include a mix of syslog, rsyslog and Syslog-NG at different version levels. This restricted the flexibility when it came to ordering the string templates for handing over to processing tools like the Hierarchical Event Log Organizer (HELO) [1]. Using Syslog-NG on the reception end allowed the greatest flexibility and we were able to find a least-common (a.k.a. most rigid) log header/message template that all the others could map to and expand upon. Syslog-NG also allows us to use it as a common transmission layer for all other typed of non-syslog data. This includes things like application logs, performance metrics, quality of service assessments, etc. Syslog-NG can handle these by using the `flags(no-parse)` option in the source definition.

Data destinations are usually forked at the Syslog-NG point into the handler and into the archive. It is easier to write twice than to traverse the entire data set later and determine what to write at that point. Archiving at reception also provides a fail-safe against data loss if something were to go wrong during ingestion processing.

A round-robin, bulk loading technique is used to move data into the database for log streams that have the potential for very high burst rates. (i.e. storms). This helps insulate the database against record injection overload. This is a common mitigation technique as indexing in bulk is far more efficient that indexing on each record inserted [2]. High rates of record inserting can overload a database engine and cause it to cease other critical functions.

B. Sources

A key feature of this method is the consideration of log messages across all of the reporting subsystems. Since there are a number of shared resources in a supercomputer, each able to become distressed independently, this ability provides a substantial benefit in perspective. The following is a list of some of these sources.

- Cray's Lightweight Log Manager (LLM): This source aggregates various system logs related to the mainframe which includes console logs from the compute and service nodes, systems logs from the supporting infrastructure machines like the System Management Workstation (SMW). Some of the subsystems included are ALPS logs, erd, xtconsumer, balanced injection, and many more.
- Bright Managed External Service Maintenance System (ESMS): This set of machine contain the login nodes, import/export (ie) nodes as well as the HPSS system nodes. Each has a separate role and the Bright Cluster Manager coordinates the centralized logging of all nodes under its domain. Fortunately, this provides a singular point from which to forward logs to the ISC.
- Sonexion Filesystems: The three filesystems supporting BlueWaters each have their own log feeds. The /home and /projects filesystems each consist of 2PiB across 144 OSTs, while the /scratch filesystem consists of 20PiB across 1440 OSTs. Logs are aggregated and forwarded to the ISC from each filesystem.
- Scheduler System: Torque and Moab each generate various logs and those are separately transferred by Syslog-NG running in parallel to rsyslog on the scheduler host, for the express purpose of application and accounting log transmission. This is required as at implementation time, the installed rsyslog service was strictly managed by LLM and unable to be expanded to accommodate these logs.
- Networking Devices: Various network switching devices with the ability to log events are also pointed toward the ISC machine to provide yet another viewpoint of what may be occurring on the system.
- Miscellaneous: Just about anything else that can generate log messages and can be configured to log remotely gets collected by the ISC.

C. Classification

At NCSA we use the in-house developed HELO system to identify and classify log messages in real time. The HELO system has been presented before, but a short recap of what it does for us is prudent since it is an enabling technology. In short, HELO is a learning system that processes log messages and identifies each message when possible and tags it with an integer identifier (TemplateID). When it is unable to match the log message to a currently known template, it analyses the current library of templates to see if there is another existing template that could be made slightly more general to include the current log message. If the generalization meets certain bounds and criteria, the template is modified and the current log message is tagged and processing continues. However, if the current log message is significantly different than any of the existing templates, it becomes a new template and the current message gets tagged with the new TemplateID and processing continues.

HELO was originally built to be run in parallel across many machines with a centralized template database. The plan of record was to handle similar log streams in a distributed manner. In the end, the architecture was still used, but each incoming log stream was directed to a separate HELO process. HELO still works in parallel, but each handles its own source. This is all accomplished currently on a single server that also functions as the ISC database server. Log message rates have continued to grow significantly and a number of optimizations have been made to HELO to continue to operate on the given hardware.

The first optimization was to dynamically reorder the template data structure in the HELO process and sort by the frequency of log message occurrence. The most frequently occurring log messages have their template moved toward the front of the list and therefore found sooner in the list traversal. By doing this dynamically, event storms get moved to the very beginning of the list and are effectively matched in constant time. In theory, using a data structure better suited to searching (O log(n)) makes more sense, and may be implemented in a later version of HELO. But in practice, given the distribution of log message frequency, this solution performs very well and adapts to the ever changing log streams.

To describe our second optimization, we must first explain one of the most trying hurdles we have had to deal with. That being log fragmentation. There have been a number of bugs across multiple sources that have resulted in the same major issue for a learning system like HELO. Essentially a log message that gets randomly split up into two or more log

messages becomes a mutating source where each fragment gets recognized as a unique log message. When all the combinations of splitting are considered for a single log message and then multiplied by the realm of possible log messages, it is easy to see how HELOs template library can grow virtually unbounded. This occurred on multiple occasions and classification times grew to a point where the classification rate could not keep up with the message rate. Our solution for this was to deactivate log messages on a basis of frequency of occurrence as well as consideration of time since last occurrence. Each HELO process now maintains a list of active templates to match against. If no direct matches are found, it then matches against the list of inactive templates on the server side. If found there, the template is re-activated. Otherwise the procedure continues as previously described. Even as the ultimate solution of correcting log stream mechanics such that fragmentation does not occur is implemented, the HELO software is now better suited to handling large event storms.

D. Message Pattern Types

In this section we will outline the different types of log message patterns that are utilized in our various analyses.

- Single Event Per Failure (Type A): This is what the rest of the non-log-processing world thinks exists to indicate a concise single point of failure. However, on systems greater than size of one, this is rarely the case. None-the-less, if and when this type of event occurs, we certainly want to identify its unique occurrence in one period and not another. But it is certainly not the only focus of our analysis. This would obviously be too trivial and uninteresting.
- Multiple Similar Event Per Failure (Type B): Expanding on the pattern presented in A, systems where multiple similarly classed components exist, it is common for many, if not each to generate an event of the same type when a failure occurs in common relevance to them. Another dimension of multiplicity is that when a failure occurs, the corresponding failure mode of a related component oftentimes will generate multiples of the same message over time as it loops or retries what it was doing when things failed. And in reality, both aspects happen such that multiple places generate multiple log messages. This amplification effect actually helps to differentiate the two compared periods and we will compare on a ratio of occurrence basis as well.
- Multiple Different Events Per Failure (Type C): Often when a failure occurs, it has a certain signature. This is characterized by a number of log messages, sometimes in a recurring order (and sometimes not) being generated. These can come from the same source component and when they do, the probability of consistent order is higher. However, they can come from multiple sources and timing can be much more loose. For example, a log message may be generated from a network link failure and that causes a shared file system error to occur and generate a message of its own. Having the ability to consider all sources at the same time is very beneficial in gaining insight as to the nature of various failure analyses. The ratios of occurrence will also help differentiate the periods and each of the multiple events will be represented.
- Constant Rate Events (Type D): There are a good number of events that occur on a constant or virtually constant basis. The quintessential example of this type of event is one initiated by a cron job. This log pattern would represent as a fairly constant heartbeat across time. When multiplied by the number of reporting components, a singular tick becomes a roaring blanket of white noise. An objective of our methods will be to eliminate this type of event even though it may be occurring with great frequency. A quick point to address an obvious rebuttal. We do not suggest that analysis of these have no value. Approaches like using signal processing techniques to these can result in interesting insights, but is outside the scope of our comparative set analysis.
- Variable Rate Events (Type E): This class of event is quite common and very hard to pin down using cursory log scanning as they will oftentimes show up in both sets. A heuristic that we consider is that certain log messages occur more frequently when things are not in an optimal state. A certain rate for a given log message may be considered normal, but a heightened rate may indicate a heightened level of distress. A human glancing at log files could likely find the event common to both and discount its relevance. However, the fact that it could occur with greater frequency is potentially an important factor. The divesture from the earlier event types is that we will need to compare rates instead of counts of occurrences.

IV. Log Processing Logistics

A. Input Data

The table we create for further analysis is dependent on both user supplied input data as well as existing HElO metadata tables.

The following is a list of input variables:

- jobset: The name of the comparison set. Used to identify the set of data versus comparisons of other sets.
- fast_start: The Unix timestamp of the beginning of the period where things ran well or normal.
- fast_end: The Unix timestamp of the end of the period where things ran well or normal.
- slow_start: The Unix timestamp of the beginning of the period where things ran poorly.

- slow_end: The Unix timestamp of the end of the period where things ran poorly.

The table with metadata that is generated as the HELO data in ingested into the database contains slightly summarized message count information. This is our our source for message data.

```
CREATE TABLE `Msg_Rates` (
`TemplateID` int(11)       NOT NULL,
`Location`   varchar(32)   DEFAULT NULL,
`EventCount` bigint(20)    NOT NULL,
`MsgTime`    bigint(20)    NOT NULL,
`Source`     varchar(16)   NOT NULL
                           DEFAULT 'unknown',
`Facility`   varchar(32)   NOT NULL
                           DEFAULT 'none',
UNIQUE KEY `TemplateID`  (`TemplateID`,
                          `Location`,
                          `MsgTime`),
KEY `MsgTime_Idx` (`MsgTime`))
```

Where

- TemplateID: The log message integer id as tagged by HELO.
- EventCount: The number of messages of type designated by TemplateID at the MsgTime.
- MsgTime: Unix timestamp.

With this info, we are then able to create our primary data table for statistical log comparison.

B. Comparison Tables

The table used to hold the statistics is defined as follows:

```
CREATE TABLE `consistency_setcounts` (
`jobset`       varchar(16) NOT NULL,
`TemplateID`   int(11)     NOT NULL,
`fast_count`   int(11)     NOT NULL,
`slow_count`   int(11)     NOT NULL,
`fast_rate`    float       NOT NULL
                           DEFAULT '0',
`slow_rate`    float       NOT NULL
                           DEFAULT '0',
`count_ratio`  float       NOT NULL
                           DEFAULT '0',
`rate_ratio`   float       NOT NULL
                           DEFAULT '0',
PRIMARY KEY (`jobset`,`TemplateID`));
```

Where

- jobset: The name of the comparison set. Used to identify the set of data versus comparisons of other sets.
- TemplateID: The log message integer id as tagged by HELO.
- fast_count: The count of the log message that exists in the period where things ran fast.
- slow_count: The count of the log message that exists in the period where things ran slow.
- fast_rate: The average rate of the log message over the period where things ran fast.
- slow_rate: The average rate of the log message over the period where things ran slow.
- count_ratio: The ratio of occurrence count.
- rate_ratio: The ratio of occurrence rates that are normalized by the length each period.

C. Method of Comparison

The first pass is to populate the table with the jobset, templateids and message counts for the first period. The following SQL statement accomplishes this.

```
INSERT INTO consistency_setcounts
  SELECT '[jobset]', TemplateID,
         sum(EventCount), 0, 0, 0, 0, 0
  FROM Msg_Rates
  WHERE MsgTime >=[fast_start]
    and MsgTime <= [fast_end]
  GROUP BY TemplateID
```

Then, for each row returned by:

```
SELECT TemplateID, sum(EventCount)
  FROM Msg_Rates
  WHERE MsgTime >=[slow_start]
    and MsgTime <=[slow_end]
  GROUP BY TemplateID
```

Do the following:

```
INSERT INTO consistency_setcounts
  (jobset, TemplateID, fast_count,
   slow_count, fast_rate, slow_rate)
  VALUES('[jobset]', [TemplateID], 0,
   [sum(EventCount)], 0, 0)
  ON DUPLICATE KEY
  UPDATE slow_count=VALUES(slow_count)
```

Next, we calculate rates and ratios with the following:

```
UPDATE consistency_setcounts
  set fast_rate =
        fast_count / [fast_end-fast_start],
      slow_rate =
        slow_count / [slow_end-slow_start]
  WHERE jobset='[jobset]'
```

and,

```
UPDATE consistency_setcounts
  set count_ratio =
    if(fast_count > 0,
       slow_count / fast_count,
       999999),
  rate_ratio =
    if(fast_rate > 0,
       slow_rate/fast_rate,
       999999)
  WHERE jobset='[jobset]'
```

V. Data Presentation

Once the processing of the log messages is complete, the mechanics of presentation is quite straight forward. It is accomplished by the following query:

TemplateID	Fast Count	Slow Count	Fast Rate	Slow Rate	Count Ratio	Rate Ratio	System	Example Message
10929	0	6	0	0.00109369	999999	999999	user	INFO: /dev/sg0 SHX0978906G07RV: 2012-12-17 19:08:48.212; IPMI; ipmi_log; 02; BMC;1;#037-0x25:08:76:0AFFFF
13223	0	3	0	0.000546846	999999	999999	local3	Request Timeout:Info1=0x804c4a100101404b:Info2=0x10005000005fd:Info3=0x7ef
6126	0	1	0	0.000182282	999999	999999	local3	sched 29s/29s/0s ago
12384	0	1	0	0.000182282	999999	999999	local3	equest Timeout:Info1=0x804c4a100101404b:Info2=0x100060000118e:Info3=0x1290
12383	0	1	0	0.000182282	999999	999999	local3	meout:Info1=0x804c4a100101404b:Info2=0x100060000193a:Info3=0x11d5
5760	0	1	0	0.000182282	999999	999999	local2	placeApp message:0x1 'claim exceeds reservation's node-count'
14629	0	1	0	0.000182282	999999	999999	local3	ago
5716	0	1	0	0.000182282	999999	999999	local2	[28174] Agent received 'Write failure to stderr of 112 bytes, ret -1'
6239	0	1	0	0.000182282	999999	999999	local3	complete_closed_conn()) Closed conn 0xffff880277383800->22922@gni (errno -110, peer errno 0): canceled 1 TX, 0/0 RD
12067	8	222	0.000336969	0.0404666	27.75	120.09	local3	2012-12-17 12:00:00 bwsmw1 45098 cb_alps_app_status: nid_to_apentry_hash contains 22572 nids
9748	1	5	4.2121e-05	0.00091141	5	21.6379	local3	:SSID Request Timeout:Info1=0x8038c3500101404b:Info2=0x10006000014c0:Info3=0x10ba
9716	1	2	4.2121e-05	0.000364564	2	8.65516	local3	Timeout:Info1=0x8038c3500101404b:Info2=0x1000500001892:Info3=0x40b
2026	592	1173	0.0249358	0.213817	1.98142	8.57471	kern	LNet: 13885:0:(gnilnd_cb.c:1116:kgnilnd_tx_done()) $$ error -11 on tx 0xffff880299fbe000->16423@gni id 1525696678/14 b00fbabe/8/10/a656/0/0 x13749:GNILND_MSG_CLOSE
10158	2	3	8.4242e-05	0.000546846	1.5	6.49137	local3	LNet: 13833:0:(gnilnd_cb.c:1116:kgnilnd_tx_done()) $$ error -11 on tx 0xffff88028e072248->16423@g
6313	23662	24280	0.996672	4.42581	1.02612	4.44059	local3	HWERR[2051]:0x0b2b:SSID Request Timeout:Info1=0x8038c3500101404b:Info2=0x1000500001bc9:Info3=0x9b9
2120	23663	23669	0.996715	4.31444	1.00025	4.32866	kern	HWERR[2051]:0x0b2b:SSID Request Timeout:Info1=0x8038c3500101404b:Info2=0x1000500001bc9:Info3=0x9b9
5753	1	1	4.2121e-05	0.000182282	1	4.32758	local2	[7171] Agent received '[NID 16507] 2012-12-17 05:53:46 Apid 244298 killed. Received node failed or halted event for nid 1
2185	2	2	8.4242e-05	0.000364564	1	4.32758	local1	[sys_sdb@34] Connected
2189	2	2	8.4242e-05	0.000364564	1	4.32758	local1	[sys_sdb@34] cb_node_unavailable: node c17-6c2s3n1 found in avail event

Figure 3: Log comparison.

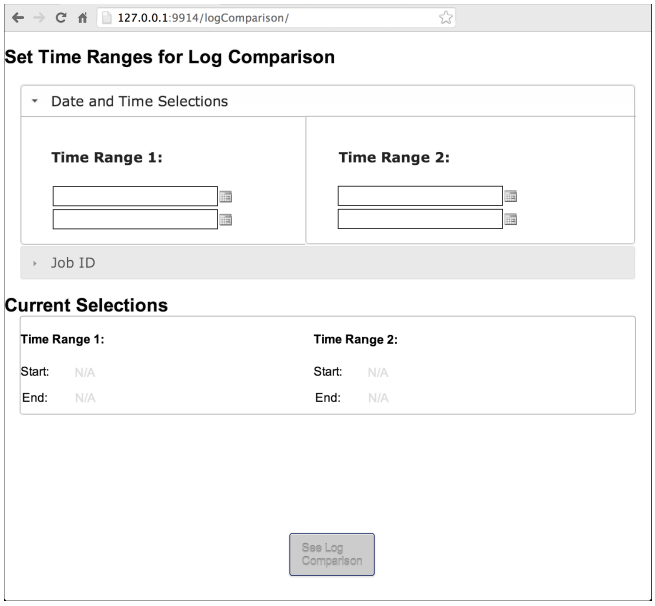

Figure 4: Time window selection utility.

```
SELECT * FROM consistency_setcounts
  WHERE jobset='[jobset]'
  ORDER BY rate_ratio DESC,
           slow_count DESC
```

This may seem fairly trivial, but during initial development, it was not. The first pass only considered total counts. This identified Log Message Patterns A, B and C with a bias for those of type C.

It was then that it was realized that not all periods of comparison would be of same or even similar length. So we normalized the counts to the number of seconds in the period. This, mathematically, is the average message rate. Then we were able to compare these rates, which enables the identification and separation of Log Message Patterns D and E. Count ratios are not necessarily useless in light of the rate ratio comparison as the counts help characterize the message occurrences. For example, log message #100 occurring 5 times in the fast period and 7 times in the slow could be significantly different than 50,000 and 70,000 respectively even though they would have the same ratios. Another example are those messages that occur only in the slow period and not in the fast period. All of these would have an infinite ratio, but those that only occurred once may carry very different weight than those that occurred many multiple times.

This leads back to a previously mentioned issue of log fragmentation. These plague the results in that there are many singular fragments that not surprisingly have one occurrence in the slow period and no match in the fast period. Therefore they show up as Log Message Pattern A, which is certainly is a false positive. Having the message counts handy help again here in identifying the possible fragment cases. Further separation occurs in the sorting by first the rate_ratio and secondarily by the slow period count. This bubbles those with a greater incidence of occurrence to the top of the list. Also, by comparing rate ratios, we have an obvious inflection point (ratio 1:1) where things are equal. However if only the count ratio was considered, we would have to know the relativity of period lengths to know where

this inflection point exists in the sorted list. This would be different for each and every comparison. But by comparing rates, this point is always at the 1:1 ratio mark.

A. Time Window Selection Considerations

The common question when a problem arises is to ask what happened then, or what was different during that period. Asking what was different implies a comparison to some norm. While it would be technically possible to consider the whole occurrence base for the recorded lifetime, far too many counter cases can be offered where invalid results would be produced due to logical changes in the system. Examples include: software upgrades where new messages that were not even possible in the past show up, changing the log reporting level of software components, adding a new log stream, even other user jobs can cause shared resources to represent different log signatures. When considering all the possible log messages, the narrowing of the basis period to as close a similar period should produce more accurate results.

This concept of "Fluctuating Normal" is characterized by the many states and degrees of those states that can be considered normal. Generally, temporally proximate periods should statistically provide the best basis for comparison in that there should be a much smaller probability that the state (or log message rates) differ due to factors other than those responsible for the observed performance difference. For this reason we have provided a simple user-facing utility to facilitate the entering of these periods of time. Figure 4 show this utility. Most importantly a user has the option to select any two periods of time for log comparison, and this is what is shown in the image. However, we expect a common case to be the desire to compare jobs and we have offered this functionality as well through the "Job ID" drop down section. Once both time ranges are adequately defined, the "See Log Comparison" button activates. Pressing this button will deploy the log comparison calculations with a result as in Figure 3.

VI. Conclusion

What we have provided is a method to process and sift through the massive amounts of log messages across multiple subsystems and present those log messages with a higher rate of incidence in a period in question. The goal is to provide a tool to aide in the identification of potential causes for various problems such as degraded job performance. This tool can also be used in diagnosing possible root causes for other observed symptoms in the maintenance of the systems health.

References

[1] J. Fullop, A. Gainaru, and J. Plutchak, "Real time analysis and event prediction engine," 2012.

[2] S. Berchtold, C. Böhm, and H.-P. Kriegel, "Improving the query performance of high-dimensional index structures by bulk load operations," in *Advances in Database TechnologyEDBT'98*. Springer, 1998, pp. 216–230.