Bridging the Gaps: Joining Information Sources with Splunk

Jon Stearley *Sophia Corwell* *Ken Lord*
Sandia National Laboratories[*]
Albuquerque, NM 87185 USA
{*jrstear,secorwe,kmlord*}*@sandia.gov*

Abstract

Supercomputers are composed of many diverse components, operated at a variety of scales, and function as a coherent whole. The resulting logs are thus diverse in format, interrelated at multiple scales, and provide evidence of faults across subsystems. When combined with system configuration information, insights on both the downstream effects and upstream causes of events can be determined. However, difficulties in joining the data and expressing complex queries slow the speed at which actionable insights can be obtained. Effectively connecting data experts and data miners faces similar hurdles. This paper describes our experience with applying the Splunk log analysis tool as a vehicle to combine both data, and people. Splunk's search language, lookups, macros, and subsearches reduce hours of tedium to seconds of simplicity, and its tags, saved searches, and dashboards offer both operational insights and collaborative vehicles.

1 Introduction and Related Work

Logs are a primary source of system debugging information, but are rife with analysis challenges [11]. This has resulted in many tools and techniques, ranging from old to new and straightforward to complicated. Since its creation in 1973, `grep` [15] has been and is still the most widely used log tool. It is simple and effective, but limited to only those tasks for which it was named: global, regular expression, print. Subsetting of data often involves time or other constraints which are difficult or impossible to express with regular expressions. Approaches range from file hierarchies (described in Section 5) to relational databases. While databases provide efficient subsetting and logical joins, expression of complex relationships requires considerable fluency with Structured Query Language (SQL), which most administrators lack. Logzilla [7] hides SQL behind a web interface and provides graphical reporting, but fails to provide logical joins. Sawmill [3] provides joins as well as graphical reporting, but involves SSQL (Sawmill SQL). In contrast, Splunk [4] provides these capabilities via a search language similar to the command syntax administrators are already fluent with. More complete listings and descriptions of tools are available elsewhere [1].

Many machine learning techniques have been applied to aid the process of extracting actionable information from logs. While a comprehensive survey of these is overdue, the focus of this paper is practical data joining, so we will provide only a brief note regarding advanced data mining. Significant effort has been applied to computer-generated but human-understandable mining rules. Methods include inductive learning [8], clustering [16], genetic algorithms [17], principal component analysis [18], and support vector machines [9]. Less interpretable results have been obtained via hidden markov models [19] and non-negative matrix factorization [14]. Independent of algorithm, data preprocessing is a necessary evil to data mining (sometimes surprisingly so!).

The first author of this paper wrote the Sisyphus log data-mining toolkit [12], which began as a research activity, and later matured in usability so that the practical effectiveness of the mining algorithm [13] could be evaluated by production administrators. New ideas require additional preprocessing and development, while also trying to maintain it as a production tool. While it possesses useful and unique features, we became aware that Splunk also possessed distinguishing and valuable features, most notably the simple and flexible combining of diverse data, and mechanisms to share knowledge among people. We thus decided to assess Splunk as a learning exercise at least, and a potential development platform at most. This paper is a first view of our findings.

We first provide a brief description of log analysis

[*]Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

in the context of supercomputers. Rather than provide a comprehensive table of log samples, we will include them as case studies throughout the paper, where we describe Splunk features including lookups, macros, and subsearches. Each section compares methodologies with and without Splunk. We then provide some comments on the needs and challenges of multidisciplinary collaboration. Finally, we provide some concluding remarks. This paper is intended neither as a tutorial nor commercial for Splunk, but rather an account of the advances it has enabled us to make on these fronts.

2 The Red Sky Supercomputer

We have deployed Splunk at Sandia National Laboratories (SNL) for supercomputers, security, and desktop systems. Its use on the latter two have enabled new correlative analysis across both our New Mexico and California sites. But the focus of this paper is on supercomputers, and although we use it on multiple such systems, we describe Red Sky as a representative case.

Red Sky is Sandias newest High Performance Computing (HPC) Linux Supercomputer built in partnership with SUN Microsystems. With 5,386 nodes having 43,088 cores, the system is designed to provide over 505 TFLOPS of capacity-class compute cycles to Sandias HPC community. At 32 KW per rack, it is the greenest supercomputer deployed at SNL to date due to energy-efficient technologies including leakage-limiting power distribution systems, Nehalem processors, and liquid-cooled rack-doors. It is also the worlds first Infiniband cluster using a 3D mesh/torus network topology. Furthermore, its center section can be physically switched among two service sections of different security levels, in order to maximize its programmatic availability.

As necessary background for future sections, we will now describe the physical hardware layout. Red Sky's 36 racks are arranged in four rows (named a-d) and nine columns (1-9). Each rack contains four chassis (1-4), each chassis has twelve slots, each slot contains two nodes, and each node has two quad-core CPUs. Each chassis also contains a chassis management module (CMM) which manages and monitors chassis components, including two Infiniband switches (a and b) each having 36 ports. The switches are connected in a 3D mesh, with loops in each direction. Each switch (and node) has an x, y, and z location in the network, upon which the routing algorithm depends.

Nodes within the system are configured to have different roles. There are administrative nodes, login nodes (where users actually log in), compute nodes (where applications actually run), I/O nodes (which provide disk access), boot nodes (the compute nodes are diskless), and gateway nodes for external access. Nodes roles, client/server relationships, physical and network locations, and other information is stored in a Genders database [6]. Node hostnames indicate their logical role, whereas switch and CMM hostnames indicate their physical location. All of these components (and others) generate logs. Our point in all this is that there are multiple layers of hierarchies in the system (we have provided only a taste), and components are managed (queried, rebooted, configured, etc) in various physical and logical groups.

Log messages describe events regarding both local and remote conditions. For instance, the message

```
_do_madrpc failed; dlid 0; 0,1,7,7,28,7,1,8,28,1,18
```

indicates that the reporting device can no longer communicate with the device at the end of the `0,1,7...` route. The route is interpreted as: transmit on port `0`, then on the next device in the network transmit on port `1`, and so on, with port `18` being the final transmission in this case. Decoding messages can be tedious, not to mention studying the distribution of events across physical or logical dimensions. These tasks are however, essential to the successful management and debugging of the system as a whole.

Supercomputers exist to run applications for users. Users submit requests for a certain number of compute nodes, having certain capabilities (memory size or access to certain filesystems for instance), for a certain amount of time. These requests are called "jobs". Distinct applications and users generally stress subsystems in distinct ways, resulting in distinct log signatures. When a node finishes a job, the scheduling subsystem may start a new job on it within the same second, which is also the time granularity of most logs. Thus, associating error messages with applications or users requires mapping to what job was running on what node at what time.

Lastly, configurations change with time. Due to the routing algorithm, the removal of a single node from the network can result in literally millions of node-to-node route changes through the system - an event which does in fact occur multiple times each day. Less frequently, software is upgraded, roles are changed, parameters tuned, and hardware is replaced. Tracing the causes of log messages as a function of these changes is extremely challenging.

We have described in broad strokes the log analysis environment which supercomputers present. As the systems and their workload are extremely valuable, it is important that administrators isolate and resolve problems efficiently.

3 Incorporating Physical Information

Every five minutes, Red Sky records the values of its many physical sensors including voltages, currents, fan

```
d4-cmm2:  PS1/V_12V | 98h | ok | 10.1 | 12.36 Volts
b1-cmm3:  PS1/S0/I_12V | A2h | ok | 10.1 | 112.50 Amps
c3-cmm4:  BL1/VPS | F1h | ok | 41.1 | 520 Watts
a6-cmm1:  PS1/FAN2/TACH | 9Dh | ok | 29.38 | 12300 RPM
d6-cmm4:  PS1/T_AMB | 95h | ok | 10.1 | 24 degrees C
```

Table 1: Sample environmental monitoring logs.

speeds, and temperatures. This results in one new file containing 11,520 lines every five minutes. Sample lines are shown in Table 1. Tokens are delimited by `|`, the first token identifies the reporting CMM and sensor, the last token indicates its value, and the middle tokens provide additional logical or physical values which we will not explain here. As mentioned previously CMM names indicate their physical location and are formatted as `RC-cmmS` where `R` is the row, `C` is the column, and `S` is the shelf number. Each CMM monitors many subsystems and each subsystem contains many sensors, identifiable as `subsystem/subsystem/.../sensor`.

These files can of course be searched with `grep`. For instance

```
grep degrees /ras/spool/sdr/2010.04.14/cmm-14.55.02 |
grep ^d6
```

shows all lines for temperatures in rack d6 on April 14 at 2:55 PM. Having to specify the exact file is tedious, let alone finding only temperatures above a certain threshold. Prior to deploying Splunk, a 654-line Perl script was written which determines the most recent file, extracts values based on command line arguments, and displays them in a (hard-coded) colorized grid corresponding to device location (row, column, shelf). It can also monitor for new files, so only one invocation is needed for ongoing by-eye monitoring (watching the output on the screen).

3.1 Setup

We then configured Splunk monitor and parse these logs, in the following manner. First, two lines were added to its `inputs.conf` file so it would monitor the correct directory:

```
[monitor:///ras/spool/sdr/]
sourcetype=sdr
```

Where `sdr` is the name we gave this type of log. Next, four lines were added to `transforms.conf` so the first word on each line would be used as the reporting host:

```
[hostfirst]
DEST_KEY = MetaData:Host
REGEX = ^(\S+):
FORMAT = host::$1
```

Where `hostfirst` is our name for the transformation rule. Finally, three lines were added to `props.conf` to use the above rule, and parse two fields of our naming (`value`, and `units`):

```
[sdr]
TRANSFORMS-host = hostfirst
EXTRACT-value = |\s+(?<value>\S+) (?<units>[^|]+)\$
```

These did require some effort to get correct, but are not particularly complex regular expressions, and are easily generalized for the other log types described in this paper. Since no timestamps appear in the messages, Splunk correctly uses the file modification time. It automatically parses many logs, but is configurable for uncommon types such as our `sdr`.

Now, the latest temperatures in rack d6 above 20 degrees are found with the command

```
splunk search "minutesago=5 degrees d6 value>20".
```

The search language assumes implicit `AND` conditions between all search criteria, explicit `AND` and `OR` can be used for arbitrarily complex logic. Splunk's web interface assumes `spunk search` so we will omit it in all future examples. A plot of the ten hottest temperatures in the system over time (see Figure 1 top) is obtained by the search

```
degrees | timechart value by host
```

Summary statistics are also easily obtainable (Figure 1 middle and bottom). Results for voltage, fan speed, etc can be similarly simply obtained. Splunk searches can auto-refresh for by-eye monitoring, and/or send notifications (including plots) upon trigger conditions (such as exceeding temperature thresholds). These are brief examples of Splunk's ease and usefulness for single sources of information.

3.2 Lookups

We are now ready to describe combining physical configuration with physical monitoring logs via lookups. Whereas Splunk can extract information from external databases, we just created a text file (based on Genders) with columns for `host`, `row`, `col`, `rack`, `shelf`, `slot`, `x`, `y`, and `z` fields. Fields are comma separated, and each line relates the values like a row in a database. By defining the lookup in `transforms.conf`:

```
[genders]
filename = genders.csv
```

these fields are now also available in searches and reports. For example,

```
degrees | lookup genders host OUTPUT shelf | stats
min(value),average(value),max(value) by shelf
```

yields shelf-wise temperature summary statistics. An explanation of the search is: find all events including the word `degrees`, for each resulting event, lookup the value of `shelf` for each `host`, then calculate the minimum, average, and maximum value within each `shelf`. In this manner, Splunk's `lookup` command performs joins with system configuration information, without SQL or extensive scripting.

The following two lines in `props.conf` tells Splunk to

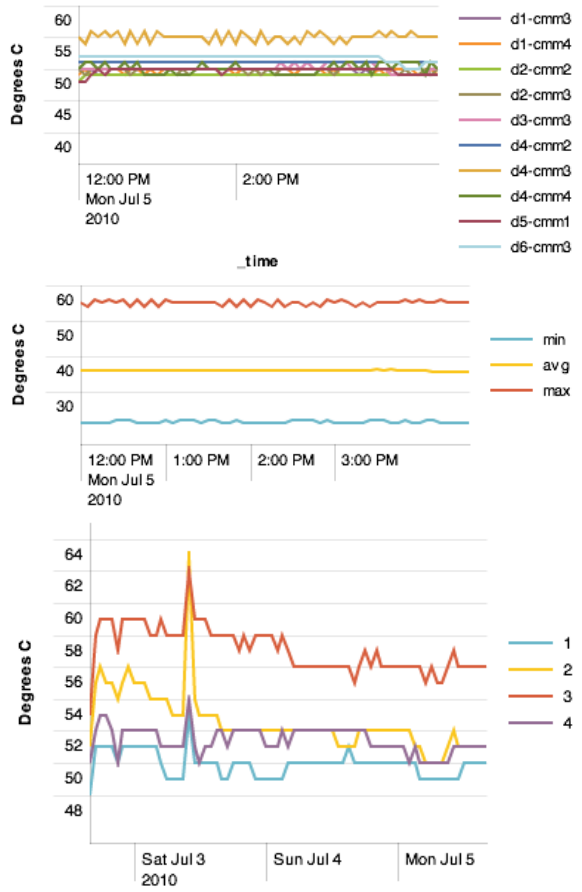

Figure 1: Temperature Plots from Splunk. Top - hottest ten reporting devices. Middle - Summary statistics. Bottom - hottest temperature per shelf. Note that plots do not share common time ranges.

perform the lookups automatically:

```
[sdr]
lookup_table = genders host OUTPUT row col rack
  shelf slot x y z
```

So the below search is now valid:

```
degrees | stats max(value) by rack
```

However, we will include the explicit lookups in examples for clarity.

We also configured Splunk to monitor a log of per-port errors on all the Infiniband network switches, and a lookup mapping routes to destinations (which we named `links`). An example of particular significance is below, which indicates bit errors in received network data:

```
Errors for 0x21283a8b120050 "b2-nem4-b" GUID
0x21283a8b120050 port 21:  [SymbolErrors == 2] Link
info:  603 21[ ] ==( 4X 10.0 Gbps Active/ LinkUp)==>
0x5080020 0008d57e4 1627 1[ ] "HCA" ( )
```

The following Splunk search produces the average number of SymbolErrors per host slot over the last 30 days:

```
daysago=30 SymbolErrors | lookup links host port
OUTPUT sender | lookup genders host AS sender OUTPUT
slot | stats avg(SymbolErrors) by slot
```

Reading from left to right, the command means: find SymbolError events within the last 30 days, for each of these lookup the host which sent the data (based on the receiving switch host and port), then lookup the slot the sender is in, and finally calculate slotwise averages of SymbolError counts. While the search is somewhat lengthy, it accomplishes in a single line a task that previously took multiple manual steps involving `grep`, `awk`, `hostspec`, `Perl`, and a plotting package. Certain slots have in fact been identified as being significantly prone to SymbolErrors, currently believed to be due to manufacturing defects.

Once a search such as the above has been written, it can be saved and shared with a few clicks, enabling reuse by the author or others.

4 Incorporating Workload Information

As described earlier, user jobs are requests for a certain number of nodes for a certain duration of time. Distinct users and applications can result in distinct log messages. Job-centric analysis of logs is a therefore a key management task. However, jobs range in size from 1 to thousands of nodes, and can last seconds to days.

We will now describe the process of attributing messages to jobs without Splunk. A syslog message of significant interest is

```
Jul 7 03:01:56 rs808 Out of memory:  Killed process
5427 (pvserver-real).
```

Out of memory errors (OOM) indicate a failure because one or more nodes assigned to the job used more than the available memory on the node. The nodes on Red Sky have 12 GB of memory, and no swap is available. The troubleshooting for these types of failures includes determining if the application is attempting to use more memory than is installed, or if there is a hardware problem on the node. However, the messages do not provide information about the job id that generated these errors, nor does it give information about the user who was running the job. The most efficient way to obtain this information is via the `sqlog -n rs808` command, which returned:

```
JOBID PART NAME USER ST START TIME N NODELIST
169357 nw pvserver fred F 04/08-20:26:47 31:33 128
rs[81-97,126-154,188-189,191-223,242-252,260-266,268-289,
790-797,808,810-813,817-836,838-843,880-887,
1170-1179,1181-1184,1186-1191,1193-1195]
```

Because the time of the OOM message is after the job started and before it ended, and rs808 is in the nodelist,

we can attribute the message to this job. While straightforward, this quickly becomes tedious, particularly for large or short jobs.

Did other nodes in the job also experience OOMs? If yes, we should follow up with `fred` regarding his memory usage, as the OOM condition is only visible from the system perspective - which he is not sufficiently privileged to view. The user-visible logs provide only the following message:

```
slurmd[rs808]:  *** JOB 169357 CANCELLED AT
2010-07-7T03:01:56 DUE TO NODE FAILURE ***
```

So whereas many nodes in the job may have OOM'd, `fred`'s logs indicate only a (single) node failure. Given this information, he is likely to believe it is a system problem (which it may or may not be), and resubmit.

On the other hand, if only one node in the job generated an OOM, we should investigate that node for problems. And so we turn to the syslogs for the answer. On RedSky, syslogs from all nodes are saved in a single file (containing 28 million lines from the last 33 days in this case), so determining if other nodes in the job had OOMs during the job is performed as follows. First, the command

```
hostspec -d '|' -w rs[81-97,126-154,...]
```

is given, which expands the hostlist into a list of nodes delimited by `|` characters (which `grep` interprets as a logical OR). Next, this is copied into the command

```
grep -E 'rs81|rs82|rs83...'  /var/log/messages | grep
"Out of memory"
```

The administrator can then scroll through the output, and examine message timestamps to determine which ones occurred during the job. The next step is to look at other messages in the job, so the second `grep` in the above is omitted, yielding many more lines to scroll through.

On another Sandia system, syslogs are instead saved into HOST/DATE files, in which case `hostspec` is used to delimit the node list with commas, and then

```
cat /logs/{rs81,rs82,...}/2010-07-0{6,7} | less
```

is used. Here the file hierarchy are used to limit the amount of extra lines that must be scrolled to (before and after the job), as the shell expands the curly brackets to only the correct hosts and only two days.

These are just the steps to view job logs. The manual process of combining with genders or other information sources is similar - command, mouse-copy, command, redirect to a file, parse with awk, pipe into other commands. Our former methods of job-centric log analysis were long and tedious.

4.1 Eventtypes, Tags, and Time-Sensitive Lookups

We define an alert as a message which warrants the attention of a system administrator. There are many types of alerts, corresponding to many types of conditions and faults within the system. Regular expressions are typically used to describe such messages, and Splunk provides this capability as well, called "eventtypes". We defined an eventtype for each known alert. We then assigned a Splunk "tag" of `alert` (our name), thereby grouping them as the set of all alerts. The simple search `tag=alert` then yields all alert messages. As before, each Splunk user can define their own eventtypes and tags, and share them with others. Eventtypes can include explanatory notes, and if additional explanation is needed the owner is visible. This is another excellent vehicle to capture and share knowledge.

By adding a time field to Splunk lookup tables, they become time sensitive and perform like a relational database with implicit time conditions. Via `cron`, we maintain a file for job lookups, containing the start time, job ID, user name, and node (one such line for each node in the job). For instance, the line

```
2010-04-08 20:26:47, 169357, fred, rs808
```

will result in lookups on `host=rs808` returning `user=fred` for events at and after that time. Corresponding end times can be set via empty values of job and user.

By using the `jobs` lookup table, messages can now be easily associated to jobs. For example, search

```
hoursago=6 tag=alert | lookup job host OUTPUT user
job | search user=fred | ctable eventtype job
```

yields a count of each eventtype for each job run by `user=fred` in the last six hours. Similarly,

```
earliest=-12hr latest=-6hr tag=alert | lookup job
host OUTPUT job user | stats count by date_hour host
job user eventtype
```

yields a table of how many of each type of alert occurred on each host, associated with the user who was running a job on the node at the time, over a six hour time window of interest. Sample output from this command is given in Table 2). Splunk has enabled us to ask harder questions in easier ways - the above search means "what is the distribution of alerts versus time, job, user, and type?" As before, subsequent lookups can be performed, results filtered or aggregate at various scales, plots generated, or results saved to files.

As a followup to the `0,1,7,7,28,7,1,8,28,1,18` message in Section 2, chained route and job lookups reveal strong correlations to certain users. The condition affects the entire system, as the network management subsystem becomes non-responsive to requests for new connections. That (non-root) users can effect such conditions indicates weaknesses in the system. Identifying the precipitating users and applications allows us to reproduce the faults at will, and drill down to the actual root of the problem. Splunk's painless decoding of such messages enables much quicker problem resolution.

hour	host	job	user	eventtype	count
10	rs1554	546324	fred	ecc:cpu:bank	36
10	rs184	599740	barney	lustre	1
11	rs1554	546324	fred	ecc:cpu:bank	37
11	rs575	546324	fred	ecc:cpu:bank	3
12	rs1554	546324	fred	ecc:cpu:bank	36
14	rs1161	602904	wilma	oom	2
14	rs1204	602904	wilma	oom	4
14	rs311	546398	betty	ecc:cpu:bank	3
15	rs1042	546716	dino	lustre	1
15	rs1043	546716	dino	lustre	1

Table 2: Search result of eventtype count per hour, host, job, and user (anonymized). Easy generation of such tables is a huge step forward (Splunk handles all the bucketing of events), enabling the next steps of examining distribution over factors of interest (time, host, and workload in this case), and identifying their root cause. Splunk provides easier ways to ask harder questions.

5 Following the Data: Subsearches and Macros

Splunk macros and subsearches further simplify the process. A subsearch is when results from one search are used as criteria for another. Just as job information is accessible via `sqlog`, it is also in logs, such as the following line from the "moab" resource management subsystem:

```
03:02:01 1278493321:3232662 job 611909 JOBEND 4
36 wilma wilma 52020 Completed [ak:1] 1278467334
1278467337 1278467337 1278493316 - - - >= 0M
>= 0M - 1278467334 64 0 -:- - FY101234 - - 0
0.00 slurm 1 0M 0M 0M 1278467334 2140000000
rs177,rs718,rs719,rs720 slurm - - [DEFAULT] - -
0.00 - - - 0,NAccessPolicy=SINGLEJOB - -
```

Where `611909` is the job id, start and end times are `1278467337` and `1278493316` (as seconds since Jan 1, 1970), running on the nodes `rs177,rs718,rs719,rs720`. We configured Splunk to monitor and parse this log, We then added the following to `macros.conf`:

```
[job(1)]
args = jobid
definition = [search sourcetype=moab job=$jobid$ |
head 1 | makemv delim="," nodes | mvexpand nodes
| eval query = " _time>=" . start . " _time<= "
. end . " " . nodes | fields + query | format
maxresults=1000]
```

The definition reads: find the most recent moab message regarding the given job id, expand the nodes field by commas, construct a new query using job start and end as time filters and job node names as content filters, and execute it (square brackets). While the construction of the macro required learning to speak Splunk, now that it is done, the simple search

```
`job(611909)`
```

yields all syslog messages from nodes in that job between start and end times - simple! We then added a Splunk "field action" (easy to do) which invokes the macro when the job field of a message is clicked on. The log review process is now:

```
tag=alert
```

and then click on a job id field in any of the resulting messages to see all the logs from the job generating that alert. As before, results can be bucketed along factors of interest, plotted, or tabled as desired. With Splunk, job-centric log analysis has become trivial!

As another example, when jobs die, administrators can run `sinfo` with various arguments to identify if any nodes are in a down state, and then go find and review the appropriate logs for clues as to why. We wrote a macro which finds the down event in the log, and then displays then preceding ten minutes of logs from that node (the number of minutes is an optional argument to the macro).

6 Collaborative Amalgam

Supercomputers are a big enterprise. They involve not only many pieces but many people, each with specialized skills and focussed priorities. While machine learning offers promise of more automated solutions, researchers with access to logs but not log experts are left to the vagaries of the logs, and are at significant risk to incorrect assumptions and interpretations [5, 11]. Conversely, administrators without new methods risk limitation to yester-year's capabilities.

Bridging these gaps presents both challenges and opportunities. By understanding the problems with todays systems, designs for tomorrows can be improved. This is true both for faults, and the fault data itself. Understanding the information quality and quantity of todays logs enables higher signal to noise in the future. While improving message quality and format is worthwhile [10], if statistical design of experiments methodology (for instance) were taken into account in supercomputer designs, perhaps the architectures themselves can facilitate the confident diagnosis of system problems. Splunk does a great job at scraping data from various sources and helping us making sense of it, but the data is of the potpourri "whatever happens to be available" flavor rather than the intentional "this data is sufficient to isolate the root cause" type.

We have described examples of how Splunk's search language, lookups, macros, and subsearch features have simplified our log analysis processes. We have also described how eventtypes, tags, and saved searches are a mechanism to capture and share knowledge. Splunk also has dashboards, which are collections of saved searches - plots, tables, event displays, whatever - and are also sharable. In addition, all searches are logged, enabling

ranking of methods based on usage (and implicitly their practical usefulness), and study of how search sessions evolve (and implicitly the driving thought processes).

Whereas we have experienced some success in these areas, there is of course room to grow. The concept of linguistic relativity [2], seems relevant regarding cross-team collaboration in general, and the adoption of Splunk for our purposes in particular. It "is the idea that differences in the way languages encode cultural and cognitive categories affect the way people think, so that speakers of different languages think and behave differently because of it." The administrators of world-class supercomputers are world-class users of the UNIX command line. They think in terms of commands invoked in parallel across sets of components, in bash, grep, awk, Perl, etc. Perceived need and interest to learn new languages and tools varies from person to person.

This is certainly also true of machine learning researchers - each has their own set of languages, skills, and tools. Different groups are used to asking different questions, and sufficiently contextualizing them to new domains is challenging. New tools must have both killer features and palatable learning curves.

7 Conclusions

Supercomputer logs are nearly as complex as the systems which generate them. We have provided examples of supercomputer logs and analysis processes, with and without Splunk. While we do not profess that Splunk solves all our analysis needs perfectly and completely, it has undoubtedly reduced hours of tedium to seconds of simplicity.

Having benefitted in operations methods, knowledge capture, and data preprocessing, we are moving to incorporate additional sources of information including routes, software versions, and configuration changes (which are version controlled, thus we have a timeline of what changed when). We are also exploring Splunk's more advanced analysis functions, and extensibility via custom search commands. Open questions include: which distributions of events over which factors (time, host, user, etc) are statistically significant, what correlations among subsystems exist, and what mining algorithms make a production impact? We want to understand the practical effectiveness of data mining techniques to help supercomputer administrators solve problems faster. In our opinion, Splunk provides both compelling features for today, and a platform for exploring new methods for tomorrow.

8 Acknowledgements

Thanks to Matthew Bohnsack, Chris Beggio, Monzy Meerza, Jerry Smith, Robert Ballance, Donna Brown, Joe Mervini, Joel Vaughan, Scott Mitchell, Randal Laviolette, and all the other experts who have contributed time, effort, and ideas to this effort! Thanks also to Dan Goldburt, Anna Tant, Stephen Sorkin, Erin Sweeney, and other Splunkers for great support and a great product!

References

[1] http://www.loganalysis.org/.

[2] *Sapir-Whorf Hypothesis*. http://en.wikipedia.org/wiki/Linguistic_relativity.

[3] *Sawmill log analysis tool*. http://www.sawmill.net/.

[4] *Splunk*. http://www.splunk.com.

[5] *Usenix Failure Data Repository*. http://cfdr.usenix.org/data.html#hpc4.

[6] CHU, A. https://computing.llnl.gov/linux/genders.html.

[7] DUKES, C. *LogZilla*. http://nms.gdd.net/index.php/LogZilla.

[8] LEE, W. Applying data mining to intrusion detection: the quest for automation, efficiency, and credibility. *SIGKDD Explor. Newsl. 4*, 2 (2002), 35–42.

[9] LIANG, Y., ZHANG, Y., XIONG, H., AND SAHOO, R. Failure prediction in ibm bluegene/l event logs. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining* (2007).

[10] MITRE. *Common Event Expression - A standard Log Language for Event Interoperability in Electronic Systems*. http://cee.mitre.org/.

[11] OLINER, A. J., AND STEARLEY, J. What supercomputers say: A study of five system logs. In *Proceedings of the 2007 International Conference on Dependable Systems and Networks (DSN)* (2007).

[12] STEARLEY, J. *Sisyphus—a log data mining toolkit*. http://www.cs.sandia.gov/sisyphus, 2008.

[13] STEARLEY, J., AND OLINER, A. J. Bad words: Finding faults in spirit's syslogs. In *Workshop on Resiliency in High-Performance Computing (Resilience)* (2008).

[14] THOMPSON, J., DREISIGMEYER, D., JONES, T., KIRBY, M., AND LADD, J. Accurate fault prediciton of bluegene/p ras logs via geometric reduction. In *1st Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2010)* (2010).

[15] THOMPSON, K. *grep - global regular expression print*. http://en.wikipedia.org/wiki/Grep, 1973.

[16] VAARANDI, R. A data clustering algorithm for mining patterns from event logs. In *Proceedings of IEEE International Workshop on IP Operations and Management (IPOM)* (October 2003), pp. 119–126.

[17] WEISS, G. M., AND HIRSH, H. Learning to predict rare events in event sequences. In *Proceedings of the 4th ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining* (1998), pp. 359–363.

[18] XU, W., HUANG, L., FOX, A., PATTERSON, D., AND JORDAN, M. I. Detecting large-scale system problems by mining console logs. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (New York, NY, USA, 2009), ACM, pp. 117–132.

[19] YAMANISHI, K., AND MARUYAMA, Y. Dynamic syslog mining for network failure monitoring. In *Proceedings of the 11th ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2005), ACM Press, pp. 499–508.