Relational Synthesis of Text and Numeric Data for Anomaly Detection on Computing System Logs

Elisabeth Baseman, Sean Blanchard
Ultrascale Systems Research Center[1]
Los Alamos National Laboratory
Los Alamos, NM 87544
Email: {lissa, seanb}@lanl.gov

Zongze Li[2], Song Fu
Department of Computer Science and Engineering
University of North Texas
Denton, TX 76203
Email: ZongzeLi2@my.unt.edu, Song.Fu@unt.edu

***Abstract*—Monitoring high performance computing systems has become increasingly difficult as researchers and system analysts face the challenge of synthesizing a wide range of monitoring information in order to detect system problems on ever larger machines. We present a method for anomaly detection on syslog data, one of the most important data streams for determining system health. Syslog messages pose a difficult question for analysis because they include a mix of structured natural language text as well as numeric values. We present an anomaly detection framework that combines graph analysis, relational learning, and kernel density estimation to detect unusual syslog messages. We design an event block detector, which finds groups of related syslog messages, to retrieve the entire section of syslog messages associated with a single anomalous line. Our novel approach successfully retrieves anomalous behaviors inserted into syslog files from a virtual machine, including messages indicating serious system problems. We also test our approach on syslog messages from the Trinity supercomputer and find that our methods do not generate significant false positives.**

I. Introduction

System logs (syslogs) are the primary way to monitor computer system health. Syslogs on desktop or laptop systems are small and manageable, but in the case of large High Performance Computers (HPC), which can consist of 20,000+ servers, the quantity of data quickly overwhelms human administrators. HPC systems typically have complex components that often interact in unexpected ways, making diagnosing incorrect system behavior a subtle and difficult problem.

System logs are an information rich set of data. Each message contains little waste text, as the purpose is to inform operators about the current status of the machine while minimizing the amount of time spent writing out logs. Each syslog message potentially contains a mixture of text and numeric data, and messages come in a variety of formats. This creates an interesting machine learning problem. We present a method for synthesizing this textual and numeric data and detecting anomalous system events.

Our work makes the following contributions:

- A novel anomaly detection framework combining relational learning, graph analysis, and kernel density estimation, to find unusual syslog messages.
- A method for detecting and grouping related syslog messages that indicate a single coherent system event.

This paper proceeds as follows: Section II reviews related work and Section III describes the syslog data our work investigates. Sections IV and V describe our approach to synthesizing textual and numeric components of syslog data, and Section VI presents our algorithm for grouping related syslog messages. Section VII describes the anomaly detection step. Section VIII details our experimental setup, Section IX presents our results, and Section X concludes.

II. Related Work

Anomaly detection on text is an open research question. Chandole *et al.* provide an excellent overview of general anomaly detection techniques [1]. Guthrie *et al.* investigate unsupervised anomaly detection on text considering only large corpora of text documents, making their work only marginally applicable to short syslog messages [2].

Kumaraswamy *et al.* note that anomaly detection on text in a specific domain is significantly improved by incorporating expert knowledge [3]. However, in the syslog domain, anomalies may be indicated by of keywords, but also can be indicated only by numeric content, such as in a kernel back trace.

Research specifically on analysis of syslog data has been sparse to date. The most notable work on analysis of textual log data was conducted by Xu *et al.* [4]–[7]. These studies focus on detecting abnormal behavior using system console logs. After parsing the logs and removing normal events, they find unusual behaviors using principal component analysis. Their goal of finding abnormal behaviors within console logs is similar to our goal of detecting anomalies in syslog data, however, they limit their study to the openings and closures of files in filesystems. We aim for a more general approach.

Other attempts at analysis of text logs include work by Fulp *et al.* in which frequencies of message signatures are investigated within sliding time windows [8]. Similarly, Gainaru *et al.* use log message frequencies to find correlated events, but do not investigate a large enough data set to draw definitive conclusions from their results [9].

[1]This work was performed at the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract DE-AC52-06NA25396. The publication has been assigned the LANL identifier LA-UR-16-25497.

[2]This work was performed during an internship at the Ultrascale Systems Research Center at Los Alamos National Laboratory in Los Alamos, NM.

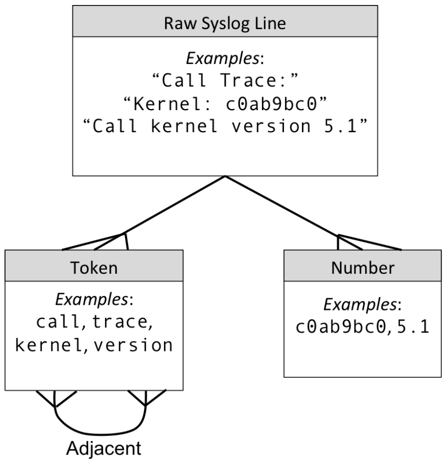

Fig. 1. The relational schema for syslog messages. We create a relational schema over the key entities in syslog messages: the raw line, the textual tokens, and the numeric data. The crow's feet indicate many-to-one relationships. Each syslog line can have zero or more textual tokens, and zero or more associated numbers. Each textual token is adjacent to 0, 1, or 2 other textual tokens. The pictured examples are fictitious simplified messages.

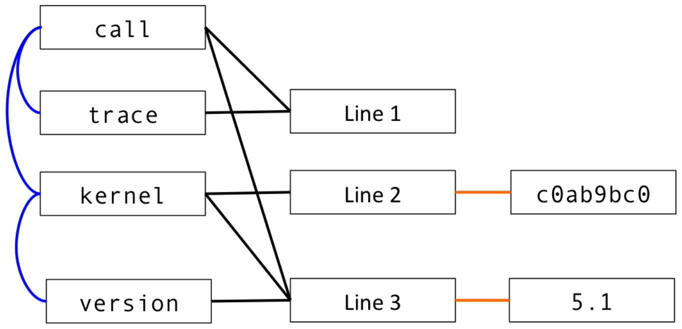

Fig. 2. The relational ground graph generated from the example messages and schema in Figure 1. One node is created for each raw line, textual token, and numeric token. Raw lines are connected to the textual and numeric tokens they contain (indicated in black and orange, respectively). Edges also connect textual tokens that occur adjacent to each other in any raw line (indicated in blue). Each textual-textual edge is annotated with the number of times the adjacency is observed in the entire syslog file.

III. System Log Data

Syslog messages are one of the most data-rich sources of information regarding system health. Information logged by the syslog utility includes a wide range of system activities. Unusual syslog messages can be indicators of serious problems, which may require human intervention. However, syslogs are long and messy, and going through them line by line by hand is time-consuming and error prone.

Machine learning can be brought to bear on syslog analysis. The syslog messages are data-rich, with content as well as structure. A basic message contains a timestamp, a prompt indicating the machine name, and the raw message content. This message can range from a single token up to about 100 characters. The message content may contain natural language text, numeric data, or a combination of the two. The natural language vocabulary of syslog is more limited than a human's vocabulary, leading to significant structure in the syslog messages. Textual data could include information about running processes and their progress, while numeric data may contain memory addresses, version information, etc.

We investigate syslog messages generated by a virtual machine as well as a subset of messages from open science testing of the Trinity supercomputer at Los Alamos National Laboratory. During this Phase 1 configuration, Trinity was a 9,436 node Cray XC30. Each node contained two Haswell Xeon E5-2698v3 16 core processors with 128 GB of main memory and a Cray Aries-based network. Syslog messages from a supercomputer of this size amount to terabytes of data per day, making automated syslog message anomaly detection crucial to monitoring and maintaining system health.

IV. Text Data: Graph Analysis

Rather than drawing on natural language processing techniques that require large corpora and assume a large vocabulary, we cast the problem of analyzing the textual component of syslog messages as a graph analysis question. This allows us to exploit the structure in the text of syslog messages. This text data is too large to enumerate efficiently, so we turn to graph clustering techniques.

Figure 1 shows syslog messages cast as a template graph, known as a relational schema, using statistical relational learning [10]. We create a node in the graph for each syslog message and each token. The crow's feet in the diagram indicate a many-to-one or many-to-many relationship. This relational schema simplifies the syslog messages by representing each type of entity (raw lines, text, and numbers) and the relationships between these entities. We add edges between textual tokens if they appear adjacent in any log lines, annotating the edges with counts of how many times the tokens occur adjacent to each other. We add edges between messages and tokens when the tokens occur within that message, and between numeric nodes and messages when the number occurs in that message. This creates an undirected, weighted graph. Figure 2 shows an example of the complete graph, known as a ground graph. This ground graph can be large, but our analysis only considers subgraphs, focusing on entities of interest.

To analyze text data in syslog messages, we focus on the subgraph of the ground graph involving only textual tokens. We use the Infomap algorithm on this subgraph to extract meaningful clusters [11]. Infomap is a hierarchical clustering algorithm based on the probability of a random walker to transition between communities in the graph as well as the probability to stay within a community.

Running Infomap on the subgraph of textual tokens gives clear, interpretable clusters. Figure 3 shows the clusters we find. Note that related terms tend to appear in the same cluster. In addition, Infomap outputs a manageable number of clusters. We use the resulting term-cluster assignments in a later step of our anomaly detection framework. Casting the problem as clustering allows our anomaly detection scheme to take advantage of content as well as structure in syslog messages.

Cluster	Top Tokens
1	activat, work, stage, device, own, woke
2	read, process, made, successful, main
3	call, trace
4	fail, no, sink-inputc, create, initial
5	all, rights, page, send, ahci, uid, cpu
6	subsequent, snd_pcm_avail, another

Fig. 3. Infomap cluster IDs based on syslog messages from a virtual machine, and their corresponding highest-ranked (stemmed and processed) textual tokens. Note that related tokens tend to be assigned to the same cluster.

V. Numeric Data: Relational Features

For each message, we extract all decimal and hexadecimal numbers. Because the format of each syslog message may differ, the representation of numbers, and indeed the count of numbers in the messages also differ. To handle this inhomogeneity, we use relational features to describe per-message numeric data [12]. Instead of including the raw numeric values in each message, we include the count of numeric values, their average, and their standard deviation. This makes us agnostic to the particular formatting of the messages. Figure 4 lists some example truncated syslog messages. After numeric feature extraction, the lines are simplified to the features shown in Figure 5. This gives, for each syslog message, a set of numeric features that function as input to the anomaly detection step of our method.

VI. Event Block Detection

A single "event" on a system, such as booting, shutting down, a failure, etc., generates one or more related syslog messages. Similar events, such as two different shutdown sequences, should be considered the same event type even though the exact syslog messages may differ. We group together related syslog messages into larger *event blocks*, assigning groups of syslog messages that indicate the same underlying event to the same event block. Our algorithm consists of three parts: (1) Pattern generation, (2) Adjacency detection, and (3) Consolidation.

Pattern Generation: When the algorithm receives a syslog message, it stores each textual token's index within the message, which we call a pattern.

Adjacency Detection: After storing a pattern for each message, we calculate the distribution over patterns. We detect patterns that always occur adjacent to each other, combining them to start building event blocks. Having done this for the entire syslog file, we obtain a list of preliminary event blocks.

Consolidation: We use our preliminary event block list to consolidate blocks tending to occur together. For each pair of blocks, we calculate the probability that the blocks occur together, within some block path length threshold. That is, if our path length threshold is 3, we allow for sequences which include a third block occurring between the two blocks being considered. For each pair of blocks A and B, we consider the probability that block A precedes block B within the given threshold, and the probability that block B precedes block A within the same threshold.

Line ID	Message Content
1	`kernel: 00000000 c0ab9bc0 00000286`
2	`kernel: sys_clock_gettime+0x98/0xb0`
3	`started daemon version 0.96`
4	`kernel: Call Trace:`
5	`Stage 5 of 5 (IP Configure) complete.`

Fig. 4. Example (truncated) syslog messages.

Line ID	Number Count	Average	StDev
1	3	1,077,490,882	1,866,268,393
2	2	164	16.971
3	1	0.96	0
4	0	0	0
5	2	5	0

Fig. 5. Example relational numeric features, calculated from the example syslog messages in Figure 4.

We set a lower bound on the probability of blocks A and B co-occurring. We found that a lower bound of 99% was sufficient to find reasonable interpretable blocks, as defined by our domain expert. In the virtual machine syslog messages, our algorithm recovers a total of 56 unique event blocks.

VII. Anomaly Detection

After extracting the Infomap clusters on textual data, and relational features on numeric data, we combine these two sets of features with keyword counts ("`error`", "`segfault`", and "`fault`") to create a single dataset. For each message, we calculate the percentage of its textual tokens contained in each Infomap cluster. We assign the message to the cluster with the maximum percentage of its tokens. We conduct one kernel density estimate per cluster to create a fine-grained anomaly detector. That is, we detect not only that a message is unusual, but that it is unusual for the *type* of message it is.

We estimate the density of each message given its cluster, ranking the messages based on this estimate, with the least dense messages ranked as most anomalous. To provide the analyst with useful context, we report the entire event block containing the anomalous message.

VIII. Experimental Setup

We run two sets of experiments: one on a set of 4,408 syslog messages from a virtual machine with unusual messages inserted, and the second on a subset of syslog messages from the Trinity supercomputer. For the dataset known to contain unusual behavior, syslog messages were extracted from a virtual machine test server using the Linux sysrq infrastructure to insert process traces into the system logs. On production machines, back traces typically occur in syslog messages when a machine encounters a strange behavior or bug. These are key in determining the root cause of a computer's bad behavior.

Preprocessing: We remove the initial set of syslog messages indicating only a boot sequence. These are uninteresting

```
kernel:[<c0709b9b>] ? ata_sff_hsm_move+0x10b/0x790
kernel: [<c04654e5>] ? irq_exit+0x35/0x70
kernel: [<c046def0>] ? process_timeout+0x0/0x10
kernel: [<c04811fb>] ? prepare_to_wait+0x5b/0x60
kernel: [<c070a220>] ? ata_sff_pio_task+0x0/0x120
kernel: [<c047b957>] ? worker_thread+0x197/0x230
kernel: [<c0480ee0>] ? autoremove_wake_function+0x0/0x40
kernel: [<c047b7c0>] ? worker_thread+0x0/0x230
kernel: [<c0480b9c>] ? kthread+0x7c/0xa0
kernel: [<c0480b20>] ? kthread+0x0/0xa0
kernel: [<c0409fbf>] ? kernel_thread_helper+0x7/0x10
kernel: ksuspend_usbd S ffffffff 0 22 2 0x00000000
kernel: f70ddaa0 00000046 ffd9da60 ffffffff 08569578 00000000 00000001 f703aacc
kernel: c0a9a580 00000000 08571efe 00000000 c0b737c0 c0b737c0 f70ddd48 c0b737c0
kernel: c0b6f0a4 c0b737c0 f70ddd48 f70fe000 bed968dd f70ddaa0 c0b737c0 c085f892
```

Fig. 6. Top-ranked anomalous syslog message event block.

messages and tend to be almost exactly the same each time. Simply running a `diff` command may be sufficient for an analyst to determine unusual boot sequences. Our event block detection scheme accurately identifies these boot sequences, automating their removal from the dataset. At some point the system shifts from setting up services to running user applications, and this is where we begin our investigation. On supercomputers this is an easily detectable state.

We pre-process the textual tokens in each syslog message. We convert tokens to the same case and remove superfluous characters, such as colons and quotation marks. A stemmer reduces each token to its root. This allows us to exploit more structure within the graph analysis. We remove tokens included in a list of irrelevant stop words, a technique in natural language processing.

Evaluation: A successful anomaly detection algorithm would recover the back traces inserted into the virtual machine syslog messages, given only the relational features we have extracted and a simple kernel density estimator. Because the Trinity data is a real set of syslog messages, we do not necessarily expect to detect any anomalous behavior. However, detecting unexpected anomalous behavior would be useful for further studies of that computing facility.

IX. Results

Figure 6 shows the event block of syslog messages ranked as most anomalous by our method run on the virtual machine server syslog file. While the top five most anomalous messages all come from this grouping, the most anomalous message we find is line 13 in Figure 6. Retrieving the event block containing that message also includes the other most anomalous lines. Presenting the analyst with the associated event block reduces time spent looking at individual anomalous messages which were generated by the same underlying event. We note that this collection of messages is a kernel back trace. This grouping is exactly the unusual behavior inserted by our analyst.

When running our method on a set of syslog messages from the Trinity supercomputer, we find that none of the individual messages have particularly low density values, and therefore all of their associated event blocks are marked by our algorithm as normal behavior. This is not unexpected, as there were no major incidents during this time period on Trinity. However, we do find this result useful in that it confirms our method does not alert on uninteresting syslog events, and as such does not produce a significant number of false positives.

X. Conclusion

We have presented an anomaly detection method for text and numeric data contained in computing system log messages. The method includes graph analysis techniques as well as relational techniques, and density estimation. We also presented a novel event block detection algorithm for finding related syslog messages that indicate a larger underlying system event. Our combined methods successfully extract unusual behavior from syslog message files, including identifying anomalous events inserted into a virtual machine server. We find no false positives when running our method on syslogs containing normal behavior from the Trinity supercomputer at Los Alamos National Laboratory. Future work includes refining more domain-specific features, and running our analyses on the event blocks themselves rather than on individual messages. Our methods are extremely useful for analysts sifting through terabytes of syslog data a day in order to diagnose and mitigate serious problems in supercomputing systems, significantly decreasing the time and effort required to identify that something has gone seriously wrong.

References

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[2] D. Guthrie, L. Guthrie, B. Allison, and Y. Wilks, "Unsupervised anomaly detection." in *IJCAI*, 2007, pp. 1624–1628.

[3] R. Kumaraswamy, A. Wazalwar, T. Khot, J. W. Shavlik, and S. Natarajan, "Anomaly detection in text: The value of domain knowledge." 2015.

[4] W. Xu, L. Huang, and M. I. Jordan, "Experience mining google's production console logs." in *SLAML*, 2010.

[5] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Mining console logs for large-scale system problem detection." *SysML*, vol. 8, pp. 4–4, 2008.

[6] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 117–132.

[7] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 588–597.

[8] E. W. Fulp, G. A. Fink, and J. N. Haack, "Predicting computer system failures using support vector machines." *WASL*, vol. 8, pp. 5–5, 2008.

[9] A. Gainaru, F. Cappello, J. Fullop, S. Trausan-Matu, and W. Kramer, "Adaptive event prediction strategy with dynamic time window for large-scale hpc systems," in *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*. ACM, 2011, p. 4.

[10] L. Getoor, *Introduction to statistical relational learning*. MIT press, 2007.

[11] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[12] J. Neville, D. Jensen, L. Friedland, and M. Hay, "Learning relational probability trees," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 625–630.