Markov Chain Modeling for Anomaly Detection in High Performance Computing System Logs

Abida Haque*
Department of Computer Science
North Carolina State University
Raleigh, NC
ahaque34@lanl.gov

Alexandra DeLucia†
Dept. of Math & Computer Science
Rollins College
Winter Park, FL
aadelucia@lanl.gov

Elisabeth Baseman
Ultrascale Systems Research Center‡
Los Alamos National Laboratory
Los Alamos, NM
lissa@lanl.gov

ABSTRACT

As high performance computing approaches the exascale era, analyzing the massive amount of monitoring data generated by supercomputers is quickly becoming intractable for human analysts. In particular, system logs, which are a crucial source of information regarding machine health and root cause analysis of problems and failures, are becoming far too large for a human to review by hand. We take a step toward mitigating this problem through mathematical modeling of textual system log data in order to automatically capture normal behavior and identify anomalous and potentially interesting log messages. We learn a Markov chain model from average case system logs and use it to generate synthetic system log data. We present a variety of evaluation metrics for scoring similarity between the synthetic logs and the real logs, thus defining and quantifying normal behavior. Then, we explore the abilities of this learned model to identify anomalous behavior by evaluating its ability to catch inserted and missing log messages. We evaluate our model and its performance on the anomaly detection task using a large set of system log files from two institutional computing clusters at Los Alamos National Laboratory. We find that while our model seems to pick up on key features of normal behavior, its ability to detect anomalies varies greatly by anomaly type and the training and test data used. Overall, we find mathematical modeling of system logs to be a promising area for further work, particularly with the goal of aiding human operators in troubleshooting tasks.

CCS CONCEPTS

• **Theory of computation → Random walks and Markov chains**; • **Computing methodologies → Anomaly detection**; • **Computer systems organization → Maintainability and maintenance**;

KEYWORDS

High Performance Computing, System Monitoring, Anomaly Detection, Machine Learning, Markov Chains

1 INTRODUCTION

Every high performance computing system requires constant monitoring and maintenance in order to ensure its health and reliability. As these supercomputing systems approach the exascale era, the sheer amount of monitoring data they generate is becoming intractable for human operators. Unfortunately, current state-of-the-art in monitoring techniques tends to be either reliance on complicated rule-based alert systems built up over years of expert experience (but which remain only able to detect previously known failure patterns), or worse, reliance simply on human operators sifting through raw data and logs by hand.

System logs ("syslogs") are one of the most crucial sources of information regarding computing system health, and indications of problems and failures, as well as hints for root cause analysis. Anomalous messages or sequences of messages within a syslog file can indicate unusual behavior ranging from unintentional user misuse or misconfigured components to serious hardware or software failures, and even malicious attacks on the supercomputing system.

In our work, we use Markov chain models, a technique from machine learning for mathematically modeling sequences of events, in order to characterize normal syslog behavior and subsequently detect unusual messages automatically. A tool using this technique would be immediately useful to senior system administrators to save time and point out previously unknown behaviors, and potentially to train less experienced system administrators.

Our main contributions are:

- Application and evaluation of Markov chain models to the system log monitoring domain for anomaly detection.
- A systematic method for evaluating system log anomaly detection schemes.

The remainder of this paper proceeds as follows: Section 2 provides a brief review of related work, and Section 3 describes the structure of syslog data. Section 4 illustrates our mathematical modeling approach in detail, while Section 5 explains our experimental setup

*This work was performed during an internship at the Ultrascale Systems Research Center, Los Alamos National Laboratory

†This work was performed during a summer internship at the Ultrascale Systems Research Center, Los Alamos National Laboratory, sponsored by the Science Undergraduate Laboratory Internships program.

‡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 AC52-06NA25396. The publication has been assigned the LANL identifier LA-UR-17-28122.

HUST'17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5130-0/17/11…$15.00
https://doi.org/10.1145/3152493.3152559

and evaluation metrics. Section 6 describes our results. Finally, Section 7 concludes and suggests future research directions.

2 RELATED WORK

Related work on mathematical modeling of system logs is sparse. Most work was performed by Xu *et al.*, on anomaly detection in system console logs, wherein they apply principal component analysis to characterize system behavior [9–12]. However, their work focuses primarily on openings and closings of files, and we attempt to tackle a much more general question. Baseman *et al.* develop an anomaly detection framework for general system logs, but do not take into account the relationships between sequences of syslog messages [2].

In addition, to date, syslog analysis tools in practice rely on detailed rules entered by human system administrators, which is error-prone and only provides the capability to identify specific previously known issues [1, 3, 6]. Our general anomaly detection scheme has the ability to identify unusual behaviors, whether or not they have been previously observed by a human operator.

3 SYSTEM LOG DATA

Syslog files can amount to millions of messages per day, with each message providing a content-rich description of the supercomputer's activities and state. While each message is generated by the kernel or some utility or daemon on the system, matching each generated message back to the piece of code that generated it is an extremely difficult task due to the large number of possible messages and variables within those messages, as well as the disorganized structure of each potentially message-generating daemon. In addition, a scheme that attempted to match each generated message back to the generating code would possibly break whenever the associated daemon was upgraded, if that upgrade included changes to the message generation code. Therefore, we cannot simply associate each message with some specific semantics and meaning extracted from the specific message-generating piece of code within some system daemon. Unfortunately, we are left having to treat each syslog message as natural language, perhaps also including some numeric values.

In general, a syslog message consists of a timestamp, a host (node) name, a tag (usually followed by a colon), and the actual message, as shown in the examples below:

Example 3.1. Jul 31 11:20:19 centostest1 acpid: starting up

Example 3.2. Jun 21 14:17:19 centostest1 kernel: [2141] 500 2141 25631 1171 0 0 0 pulseaudio

The tag indicates the kernel, a daemon, or some system utility that generated the message, and the message itself involves a combination of natural language and numeric values. In this work, we focus solely on the sequences of message types, so we remove numeric values from the message and focus on the cleaned message and the associated tag. Likewise, we treat time as dimensionless, only used as an index to provide an ordering on messages.

In this work, we specifically investigate syslog files from Grizzly and Wolf, two large institutional computing clusters at Los Alamos National Laboratory. Over the time periods we collect, our system administrators advise us that the behavior of both clusters was relatively uninteresting. Therefore, an accurate mathematical model of the behavior of these clusters should be a decent approximation of general normal syslog behavior. Due to the intractability of extracting specific semantics and meaning from each syslog message by tracing it back to its generating code, we turn to probabilistic machine learning techniques that are uniquely suited to this problem.

4 APPROACH

Our approach to anomaly detection involves using a Markov chain model, a technique from machine learning, to learn patterns of normal syslog behavior. Once we have a learned probabilistic model of normal behavior, we can then compare it against observed syslog messages in order to detect anomalous behavior. Essentially, a learned model of normal behavior allows us to estimate the probability of each observed syslog messages, given the previously observed sequence of messages. Then, we call the messages with the lowest estimated probabilities anomalous and alert a system administrator to further investigate.

4.1 Learning a Model of Normal Behavior

Markov chain models are based on learning a *transition matrix* from training data [5]. In this case, our training data is a large set of syslog files which we believe to contain almost entirely normal behavior. Transitions refer to a sequence of only two syslog messages (a transition from one message to the following message in time).

After the cleaning process described in Section 3, we are left with a finite set of generic syslog messages. Thus, the entire training syslog file can be represented as a sequence of messages, $\mathbf{X} = \{X_1, ..., X_n\}$. Then, the associated transition matrix contains elements $P_{ij} = P(X_{k+1} = j | X_k = i)$. We can estimate these probabilities by calculating them based on our training set of syslog messages, which gives us the maximum likelihood estimate of the "real" transition matrix. Note that it would be impossible to measure the "real" transition matrix because we would have to have complete knowledge of all possible syslog message sequences as well as their relative frequencies. We build our transition matrix following the algorithm presented by Teodorescu, except where general syslog messages correspond to Teodorescu's states [7]:

(1) Let n_i be the number of times message i is observed in the sequence $\{X_1, ..., X_{n-1}\}$ (exclude the last message).
(2) Let n_{ij} be the number of times we see message i go to message j. For the last message in the list, it might be the case that it only appears that one time. In that case, set $\hat{P}_{ii} = 1$.
(3) Then $\hat{P}_{ij} = \frac{n_{ij}}{n_i}$

Note that since we can only measure messages we observe, there is no chance of n_i being 0, except for possibly the last message in the list.

In this way, we arrive at a transition matrix where each element $P_{i,j}$ is the maximum likelihood estimate for the probability that syslog message j directly follows syslog message i.

4.2 Verification of the Model

After learning the transition matrix from a training set of syslog messages representing normal behavior, we would like to verify

that the learned model has indeed picked up the patterns of normal behavior. In order to do this, we can use the learned transition matrix to generate a synthetic syslog message sequence and compare it to the sequence observed in the training data. If the synthetic and real message sequences are similar, we can conclude that we have learned a model of normal syslog behavior, and we will move on to evaluating its usefulness for anomaly detection.

4.3 Generating a Synthetic Normal Syslog

In order to generate a synthetic syslog with normal behavior, we first need to select compute nodes that we expect to show normal behavior. On the two clusters of interest, Wolf and Grizzly, the first and second nodes are reserved for system administrators and tend to show fairly normal behaviors. For Wolf, these are nodes wf001 and wf002; for Grizzly they are gr0001 and gr0002. For each of these nodes, we learn a transition matrix as previously described.

Once we have the learned transition matrices, we can use each to generate a synthetic syslog that we expect to depict normal behavior by taking a random walk across the transition matrix [5]. We pick a starting message as the initial location for the random walker, and let the random walker move between syslog messages according to the learned probabilities at each transition matrix entry the walker visits. By recording the messages visited by the random walker in order we can generate a synthetic syslog. We then compare the generated syslog to the normal syslog that we used for training. If the two are similar, we can conclude that we have successfully learned patterns of normal syslog behavior.

Figure 1 is a visualization of the real syslog messages from the wf001 over time, while Figure 2 is a visualization of the synthetic syslog message sequence generated by a random walker on the transition matrix learned from wf001. The x-axis axis shows time, while the y-axis represents an enumeration of the general syslog messages (and so ordering should be ignored). The message occurrences have been colored to denote messages with similar content [8]. Note that both the real and synthetic message sequences appear to exhibit the same few states, in a similar order (the time spent in each state is different, but in this work we treat time as dimensionless so this is not a concern). Both message sequences begin fairly stable, change to a mixture state of a variety of messages, change to a fairly stable state of the same two or three messages, change again to a longer period of a mix of messages, and then settle on a long final state of transitions between only three different messages. Due to the similarities between the real and synthetic syslogs, we conclude that we have in fact learned a model that represents normal behavior. We now proceed to evaluate its usefulness for anomaly detection.

5 EXPERIMENTAL SETUP

Given a transition matrix learned from a training sequence of syslog messages we can now use that transition matrix to estimate the anomalousness of messages in any syslog message sequence. We call the syslog on which we want to do the actual anomaly detection the *test* syslog. For the each message transition in test syslog, we can use the learned transition matrix to estimate the probability of that transition. Then, ranking the test syslog messages by increasing probability will return the most anomalous messages.

In order to conduct controlled experiments using this modeling technique, we insert anomalies into our test syslogs by hand so that we know in advance which messages the model should be identifying. We insert two different kinds of anomalies: insertions of a single unusual message (which should result in the unusual message being returned, as well as the message preceding it), and deletions of a single normal message (which should result in the message preceding it being returned). In this way, an insertion anomaly actually creates two anomalous messages in the eyes of the model, while a deletion creates only one anomalous message. We evaluate the ability of the model to identify these two types of anomalous behavior, as well as the effect of different training and test sets on the performance of the model.

For our possible datasets, we extract seven separate days of syslog messages from the Wolf cluster, and seven separate weeks of syslog messages from the Grizzly cluster. This gives us 14 separate datasets or varying similarity, for a total of 49 different possible train/test pairings. Note that these pairings vary in difficulty for the task of anomaly detection, because some pairings come from the same machine over the same time interval (which would be the easiest task), whereas some pairings come from two different machines altogether (which would be the hardest task). We evaluate the performance of the model on the anomaly detection task for each of the possible train/test pairings, and for each pairing, on only insertion anomalies, only deletion anomalies, and a combination of insertion and deletion anomalies.

We also investigate the effect of our choice of scoring mechanism on the ability of the model to identify anomalies. We consider two types of scoring mechanisms: (1) simply using the estimated probability of transitions between individual message types for ranking, and (2) Using a combination of transition probabilities between individual messages and the probability of transitioning between two tags for ranking. We can build a tag transition matrix in the same way as we build a message transition matrix, and add the tag transition and message transition probabilities to obtain a score that takes into account the broader context of the messages. Therefore, for each train/test pair and for each set of anomaly types, we also test each of the two scoring methods.

5.1 Evaluation Methods

To evaluate the model's performance on the anomaly detection task, we have created a variety of anomalies by hand and therefore know exactly which syslog messages a model with perfect performance would identify. After applying the model and obtaining a ranking of the test syslog messages, we can evaluate how well the model performed by looking at where each of the created anomalies occur within the returned ranking, through calculating *precision* and *recall* values [4].

For our task, we calculate precision at 10 for each experiment. This essentially gives us the percentage of highly ranked messages that were correctly identified as anomalies. Likewise, we measure recall at R, where R is the total number of created anomalies in each experiment. This means that recall measures the percentage of created anomalies that were correctly identified as anomalous. For both precision and recall, 0.0 is poor performance, while 1.0 is perfect performance. Considering both precision and recall gives

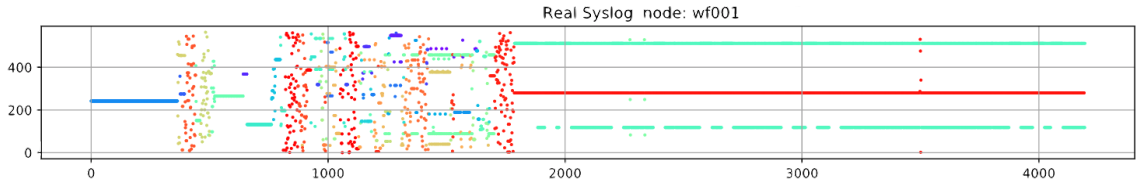

Figure 1: A visualization of enumerated general syslog messages from node wf001, exhibiting normal behavior, over time. Note the readily apparent multiple behavior states.

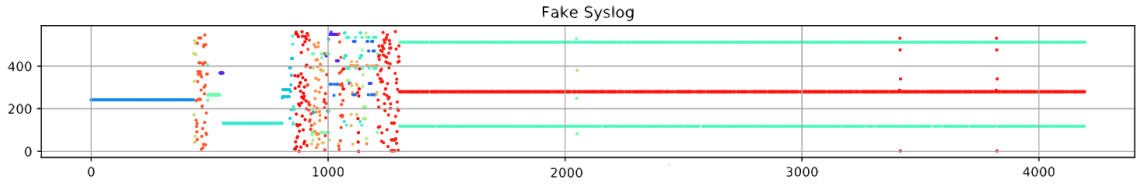

Figure 2: A visualization of synthetic enumerated general syslog messages generated by a random walker on the transition matrix learned from Figure 1. Note the strikingly similar behavior patterns.

us a normalized way to compare the performance of model in all of our experimental settings.

For further detail, precision and recall are mathematically defined as follows:

$$\texttt{Precision} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FP}} \tag{1}$$

$$\texttt{Recall} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FN}} \tag{2}$$

Where TP ("True Positives") is the number of created anomalies that appear in the top 10 or R ranks, FP ("False Positives") is the number of messages in the top 10 ranks that are not created anomalies, and FN ("False Negatives") is the number of created anomalies that do not appear in the top R ranks. For each experimental setup, we report the average and standard deviation of both precision at 10 and recall.

6 RESULTS

6.1 Ranking Based on Message Transition Probabilities Only

6.1.1 Insertion Anomalies. Figure 3 shows precision and recall results for insertion anomalies when ranking based only on message transition probabilities. The colors indicate similarity between train and test sets. We see that identifying insertion anomalies appears to be a hard problem, as precision remains around 0.5 even on the easiest train/test set pairings. Meanwhile, recall provides much better results, likely because the inserted transitions were not unlikely enough to merit a ranking within the top 10, but were still anomalous enough to end up fairly close to the top of the rankings. In general, this suggests that extra syslog lines may be hard to detect because the inserted lines must have low transition probabilities with their neighbors, and this may not be the case in general. We also note, as expected, a strong dependence on the difficulty of the train/test set pairing, with a very steep drop off on train/set sets that are draw from different clusters.

6.1.2 Deletion Anomalies. Figure 4 shows precision and recall results for deletion anomalies when ranking based only on message transition probabilities. The colors indicate similarity between train and test sets. We note that identifying missing syslog lines appears to be an easier task than identifying insertions, with both precision and recall achieving values above 0.75, even on train/test set pairings drawn from different nodes on the same cluster. Again, we see a sharp drop in performance when applying a model trained on one cluster to data from a different cluster. However, performance is good on all other train/test pairings, whether they be from the same cluster, node, and time period, different time periods, or even different nodes. Note that creating 10 deletion anomalies results in precision at 10 being identical to recall at R.

6.1.3 Insertion and Deletion Anomalies. Figure 5 shows precision and recall results for a combination of insertion and deletion anomalies when ranking based only on message transition probabilities. The colors indicate similarity between train and test sets. Including insertions anomalies along with deletion anomalies again complicates the anomaly detection task for the same reasons as described above for insertion anomalies. We observe mediocre precision results along with some quite good recall results (including some very good results with lower standard deviation). Again, we see the strong dependence on train/test set similarity.

6.2 Ranking Based on Message and Tag Transition Probabilities

6.2.1 Insertion Anomalies. Figure 6 shows precision and recall results for insertion anomalies when ranking based on the combination of tag and message transition probabilities. The colors indicate similarity between train and test sets. We note that for both precision and recall, the trials that achieve exceptionally high performance also have significantly decreased standard deviations. In addition, the lower bound on precision and recall for each type of train/test pair seems to have increased slightly. In particular,

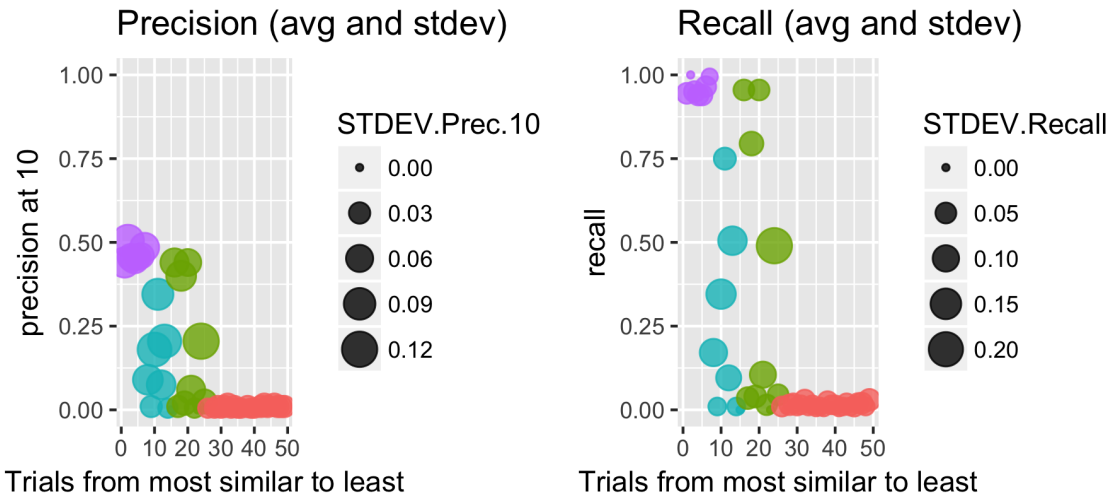

Figure 3: Precision and recall results for insertion anomalies when ranking based only on message transition probabilities. The colors indicate similarity between train and test sets: train and test sets are taken from the same cluster, same node, and over the same time period, train and test sets are taken from the same cluster, same node, but over different non-overlapping time periods, train and test sets are taken from the same cluster, but different nodes, and train and test sets are taken from different clusters.

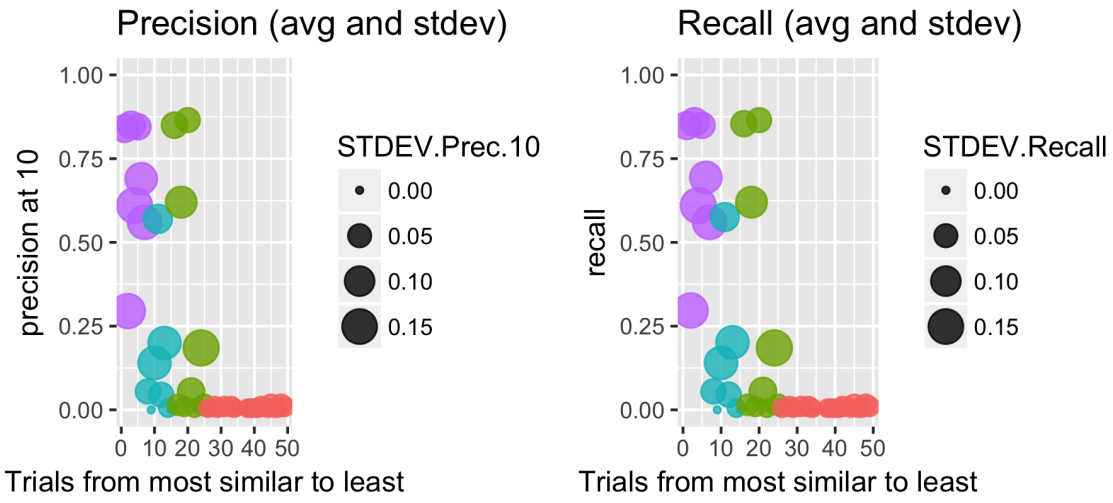

Figure 4: Precision and recall results for deletion anomalies when ranking based only on message transition probabilities. The colors indicate similarity between train and test sets: train and test sets are taken from the same cluster, same node, and over the same time period, train and test sets are taken from the same cluster, same node, but over different non-overlapping time periods, train and test sets are taken from the same cluster, but different nodes, and train and test sets are taken from different clusters.

train/test pairs that 0.0 precision or recall using only message transition probabilities now manage to find at least some correct answers. This suggests that including the tag context of the messages helps the model to identify unusual inserted lines.

6.2.2 Deletion Anomalies. Figure 7 shows precision and recall results for deletion anomalies when ranking based on the combination of tag and message transition probabilities. The colors indicate similarity between train and test sets. Again, note that creating 10 deletion anomalies means that precision at 10 and recall at R are identical. We see the same strong dependence on train/test similarity, particularly with regard to train/test pairings drawn from

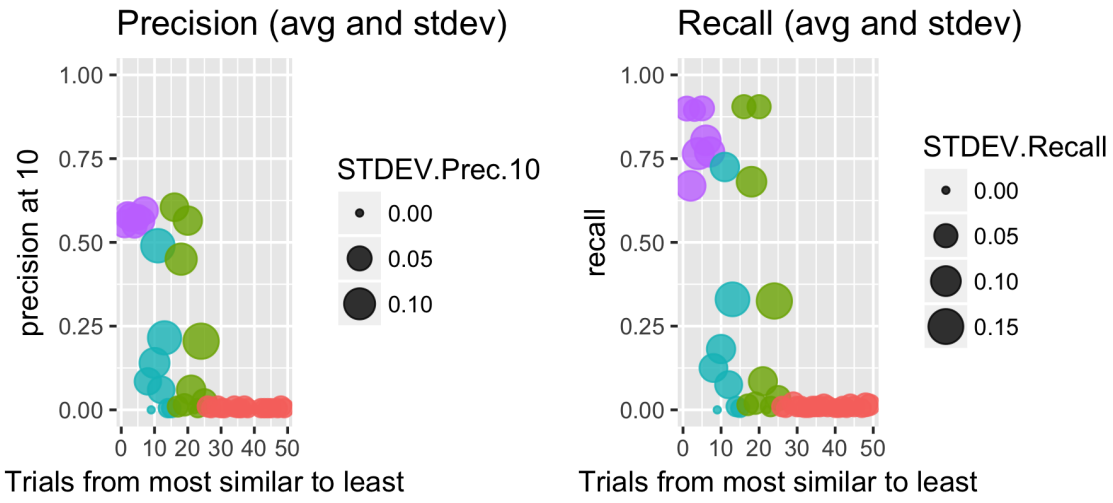

Figure 5: Precision and recall results for a combination of insertion and deletion anomalies when ranking based only on message transition probabilities. The colors indicate similarity between train and test sets: train and test sets are taken from the same cluster, same node, and over the same time period, train and test sets are taken from the same cluster, same node, but over different non-overlapping time periods, train and test sets are taken from the same cluster, but different nodes, and train and test sets are taken from different clusters.

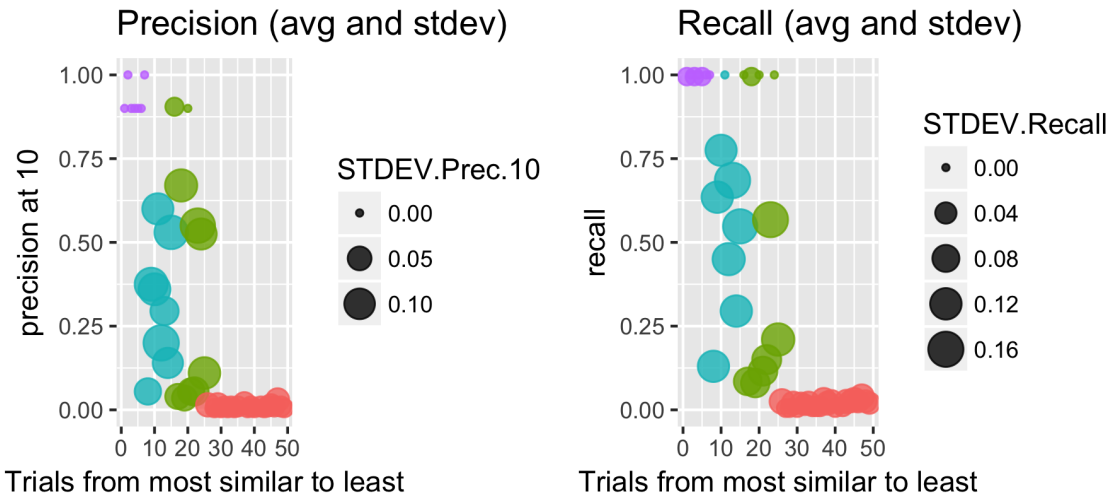

Figure 6: Precision and recall results for insertion anomalies when ranking based on the combination of tag and message transition probabilities. The colors indicate similarity between train and test sets: train and test sets are taken from the same cluster, same node, and over the same time period, train and test sets are taken from the same cluster, same node, but over different non-overlapping time periods, train and test sets are taken from the same cluster, but different nodes, and train and test sets are taken from different clusters.

different clusters, but again that the lower bound on our metrics seems to have increased with the incorporation of tag context.

6.2.3 Insertion and Deletion Anomalies. Figure 8 shows precision and recall results for a combination of insertion and deletion anomalies when ranking based on the combination of tag and message transition probabilities. The colors indicate similarity between train and test sets. We observe the same trends as in previous results, including good precision and recall results as long as the train and test sets are drawn from the same cluster. In general, including tag context seems to increase the performance of the model.

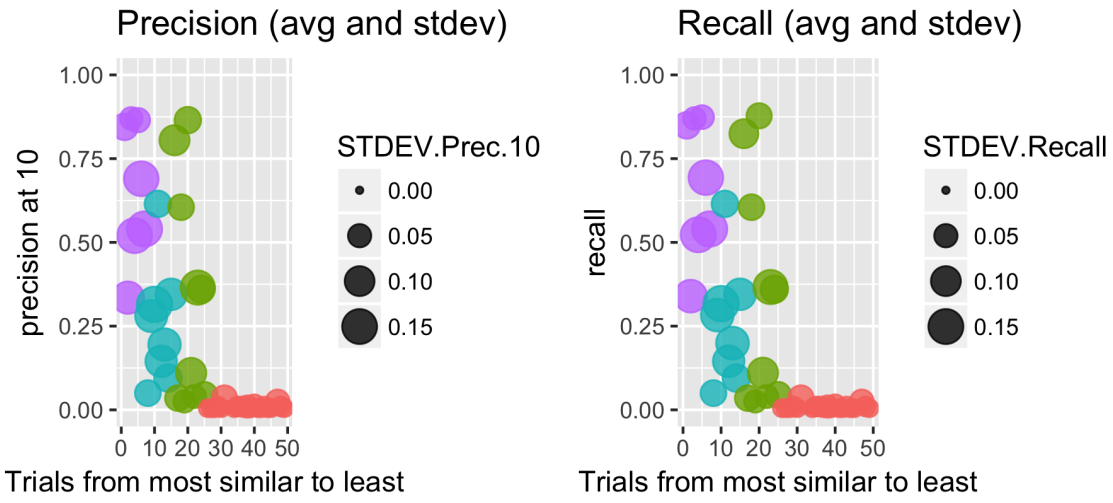

Figure 7: Precision and recall results for deletion anomalies when ranking based on the combination of tag and message transition probabilities. The colors indicate similarity between train and test sets: train and test sets are taken from the same cluster, same node, and over the same time period, train and test sets are taken from the same cluster, same node, but over different non-overlapping time periods, train and test sets are taken from the same cluster, but different nodes, and train and test sets are taken from different clusters.

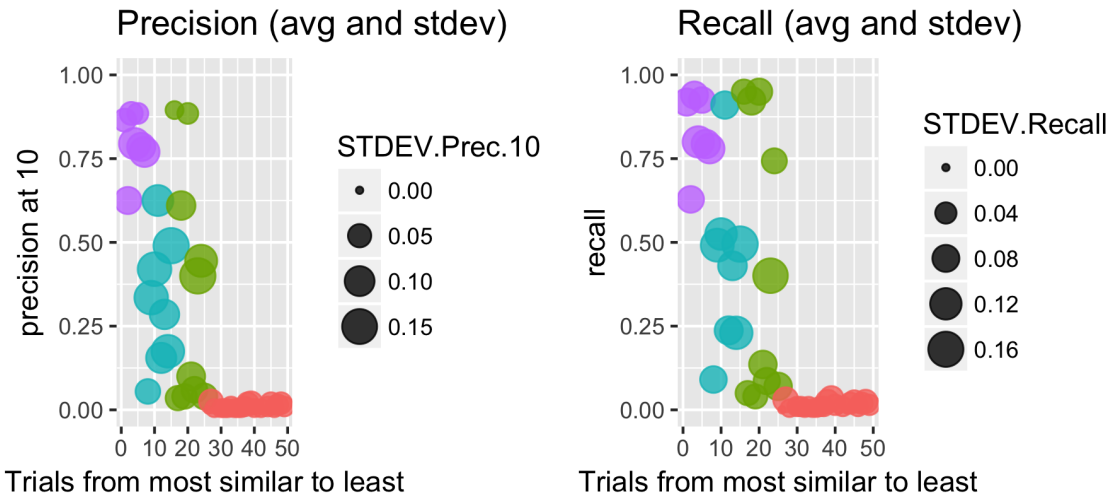

Figure 8: Precision and recall results for a combination of insertion and deletion anomalies when ranking based on the combination of tag and message transition probabilities. The colors indicate similarity between train and test sets: train and test sets are taken from the same cluster, same node, and over the same time period, train and test sets are taken from the same cluster, same node, but over different non-overlapping time periods, train and test sets are taken from the same cluster, but different nodes, and train and test sets are taken from different clusters.

7 CONCLUSIONS

We present an application of Markov chain modeling to the problem of anomaly detection within sequences of high performance computing system log messages, as well as two possible schemes for scoring and ranking system log messages in terms of anomalousness based on the learned probabilistic model. We measure the performance of the learned models on the anomaly detection task through precision and recall metrics, and investigate the effects of differences in train and test sets, types of anomalies, and ranking schemes on performance. We find that, in general, Markov chain modeling is an effective method for detection of unusual system log message sequences. In particular, we note the expected strong dependence on similarity of train and test sets. However, performance only decreases severely if the train and test sets are

drawn from completely different computing clusters; the method still works with reasonable precision and recall even if the train and test sets are drawn from different compute nodes within the same compute cluster. We observe that using the probabilistic model to rank messages based also on the probability of context tags within the message, rather than only the message itself, improves model performance, especially in the case of inserted anomalies. Overall, we believe that Markov chain modeling of system log sequences is a promising area for anomaly detection and deserves further investigation, and perhaps refinement of the model to improve performance.

Future work includes incorporating more nuanced notions of message context, such as labeling message context based on unsupervised clustering models in addition to using the provided tags. In addition, we plan to incorporate more detailed ideas of time in our analysis. In this work, we only consider time as a dimensionless ordering index. We plan to expand the model so that instead of considering the messages as a discrete time series, we consider it as a point process, taking into account messages which tend to occur close together in time or almost simultaneously. We expect that the incorporation of these additional features will improve the model performance on the more disparate train/test set pairings.

Ultimately, after further work to improve model performance on both the insertion and deletion anomaly detection tasks, we anticipate developing the model into a tool for system analysts to assist in failure detection, troubleshooting, and root cause analysis.

ACKNOWLEDGMENTS

The authors thank Sean Blanchard, at the Ultrascale Systems Research Center, Los Alamos National Laboratory, for crucial domain expert input, and Nathan DeBardeleben, director of the Ultrascale Systems Research Center for organizational support.

REFERENCES

[1] Paessler AG. 2017. Syslog Monitoring: Effectively Manage Your System Messages. (2017). https://www.paessler.com/syslog-monitoring

[2] Elisabeth Baseman, Sean Blanchard, Zongze Li, and Song Fu. 2016. Relational Synthesis of Text and Numeric Data for Anomaly Detection on Computing System Logs. In *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*. IEEE, 882–885.

[3] Zoho Corp. 2017. Syslog Monitoring. (2017). https://www.manageengine.com/network-monitoring/syslog-monitoring.html

[4] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 233–240.

[5] Kevin P Murphy. 2012. *Machine learning: a probabilistic perspective*. MIT press.

[6] LLC Nagios Enterprises. 2017. Syslog Monitoring with Nagios. (2017). https://www.nagios.com/solutions/syslog-monitoring/

[7] Iuliana Teodorescu. 2009. Maximum likelihood estimation for Markov Chains. *arXiv preprint arXiv:0905.4131* (2009), 4–6.

[8] Stijn Marinus Van Dongen. 2001. *Graph clustering by flow simulation*. Ph.D. Dissertation.

[9] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009a. Online system problem detection by mining patterns of console logs. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, 588–597.

[10] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009b. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 117–132.

[11] Wei Xu, Ling Huang, Armando Fox, David A Patterson, and Michael I Jordan. 2008. Mining Console Logs for Large-Scale System Problem Detection. *SysML* 8 (2008), 4–4.

[12] Wei Xu, Ling Huang, and Michael I Jordan. 2010. Experience Mining Google's Production Console Logs.. In *SLAML*.