Predicting Job Completion Times Using System Logs in Supercomputing Clusters

Xin Chen*, Charng-Da Lu† and Karthik Pattabiraman*
*Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T1Z4, Canada.
Email: {*xinchen, karthikp*}*@ece.ubc.ca*
†Buffalo, NY 14214, USA.
Email: *charngdalu@yahoo.com*

Abstract—**Most large systems such as HPC/cloud computing clusters and data centers are built from commercial off-the-shelf components. System logs are usually the main source of choice to gain insights into the system issues. Therefore, mining logs to diagnose anomalies has been an active research area. Due to the lack of organization and semantic consistency in commodity PC clusters' logs, what constitutes a fault or an error is subjective and thus building an automatic failure prediction model from log messages is hard. In this paper we sidestep the difficulty by asking a different question: Given the concomitant system log messages of a running job, can we predict the job's remaining time? We adopt Hidden Markov Model (HMM) coupled with frequency analysis to achieve this. Our HMM approach can predict 75% of jobs' remaining times with an error of less than 200 seconds.**

Keywords: Log Analysis, Prediction, Hidden Markov Model

I. Introduction

Supercomputing systems often consist of large numbers of commodity nodes to facilitate high-performance applications. These systems use batch job schedulers to accommodate a large number of users with different computational and resource requirements. However, these systems also experience high rates of failures, due to their large scale and complexity of hardware and software interactions. One of the main challenges in supercomputing systems is to ensure that applications continue to achieve high throughput and complete the task despite hardware and software failures [1]. Therefore, it is beneficial for the job scheduler to predict every job's status and use this information to make better workload management decisions, give users feedback about their jobs' completion time, and perform proactive fault-management actions such as system-initiated checkpointing.

A job is characterized by the requested resources: usually running time and number of nodes needed. The allotted resources are mostly reserved exclusively for that job. A basic batch job scheduler maintains job queues and executes jobs on a first-come-first-serve basis. Ofttimes a technique called "backfilling" [2], which depends greatly on accurate job runtime estimation, is adopted. Backfilling lets small jobs to move ahead in queue to use nodes which otherwise remain idle. In practice, many users have very poor estimation of their job duration, causing inefficient backfilling job allocation and overestimated queueing time prediction.

Moreover, if a job runs on a resource that becomes unavailale, e.g. due to software hiccups or hardware failures, before finishing the job, then the job has to be suspended a priori and resumed later. In the past, this has been achieved through the use of checkpointing and recovery [3], which are reactive techniques. However, checkpointing and recovery alone are insufficient to ensure high throughput [4], and need to be complemented with pro-active job runtime prediction [5]. To be effective, a prediction technique should (i) predict job termination with sufficient lead-time for the application or system to take preventive action, and (ii) have low rates of false-positives (e.g., system predicts a job finishes in five minutes but it does not). The prediction should be based on a number of indicators about a system's health, such as system logs, temperature, and workload measures.

In this paper, we propose a prediction technique for supercomputing systems that makes use of the system logs to predict job residual times. System logs are free-form text messages recorded by the Linux operating system's "syslog" facility [6]. The operating system kernel, device drivers, daemon processes, and user applications all can submit message to "syslog" whenever an event occurs. An example of such a message is "hello_world[32235]: segfault at 3fffffffd ip 000000375ca7a9cc". System logs are produced during normal operation, and hence the mere presence of log messages does not indicate an error. Therefore, the main challenge is to isolate system log messages that are indicative of job termination, and use them to predict failures in the future.

We are not the first to use system log messages for prediction in supercomputing systems. For example, Oliner et al. [7] analyze failures in five supercomputing systems based on alerts that are annotated by system administrators. Liang et al. [8] use tagged logs from the BlueGene machine to discover correlations between fatal and non-fatal events, and thus predict failures. Our approach differs from these approaches in that we start with *untagged* system logs and do not require any hints or annotations from the system administrators. By not requiring such tags or annotations, our approach can potentially handle a more diverse and larger volume of system logs than prior approaches.

The dataset in this study comes from the "Edge" cluster at the State University of New York at Buffalo and comprises one month of data across about one thousand compute nodes.

There are no annotations or tags on the data, and all that we have available in addition to the logs themselves are the job start and finish times, allotted nodes, and user IDs for the job. We use *Hidden Markov Models (HMMs)* to learn the characteristics of log messages and use them to predict job residual times. An HMM is a Markov model in which the system being modelled is assumed to be a Markov process with hidden states and transitions that are not directly visible, but have outputs that are visible. HMMs have been successfully used in speech, handwriting and gesture recognition. HMMs are well suited to our domain as we have the observations of the system in the form of log messages and job completion times, but no knowledge of the internal state of the system, which is "hidden".

Our main contributions are summarized as follows:

- Identify the job residual time as an important and feasible target for prediction
- Quantify the relative importance of a log message in residual time prediction
- Build HMMs from the log messages to predict the job residual time
- Evaluate the accuracy of the HMMs in predicting residual times of different types of jobs. We find that the HMMs accurately predict the residual times for about 75% of the jobs within 200 seconds. We can further improve the accuracy to 93% if we train the HMMs with only the short jobs (i.e., jobs that run for one hour or less).

The rest of the paper is organized as follows. In section II we gives more details on HMMs, log messages, and related work. Section III elaborates our HMM-based residual time prediction algorithm. Section IV discusses experimental results and analysis. The paper concludes in Section V with suggestions for future work.

II. Background

In this section, we present background material on Hidden Markov Models, the logs used in our study, and related work on log analysis.

A. Hidden Markov Models (HMMs)

An HMM is a Markov model in which the system is driven by hidden states and observable variables depend on the states [9]. The transition probabilities between hidden states are similar to those in Markov chains, but each state has an emission probability distribution over all outputs in the HMM. Figure 1 gives an example of HMM. It is characterized by the following modules: hidden states $X = \{x_1, x_2, x_3\}$, observations $Y = \{y_1, y_2, y_3\}$, transition probabilities $A = \{a_{ij}\} = \{P[q_{t+1} = x_j | q_t = x_i]\}$, output probabilities $B = \{b_{ik}\} = \{P[q_t = y_k | q_t = x_i]\}$, and initial state probabilities $\pi = \{\pi_i\} = \{P[q_1 = x_i]\}$. The q_t's are the time sequence of states/observations. Collectively the parameter set of an HMM is denoted as λ=(A,B,π).

In this study, we calculate the transition probabilities A and emission probabilities B from the training data. We then find the optimal hidden state sequence for the observed sequence

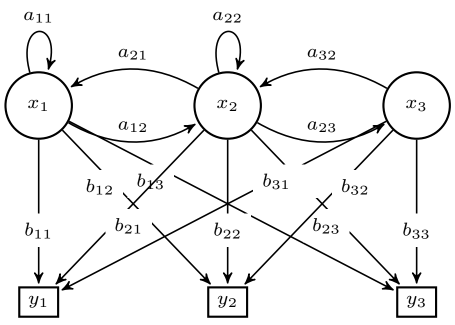


Fig. 1. An example of HMM

using the Viterbi algorithm [9], and predict the job residual times using the absorption time to the final state of the HMM.

B. Supercomputing System Logs

The system logs we used are collected from the x86-based supercomputing cluster "Edge" at SUNY at Buffalo. The cluster runs the Linux OS and manages workload via the PBS batch job scheduler. The schedule takes user job scripts, allocates compute nodes, and executes the specified commands/tasks unattended until either the job completes or the requested time expires.

Our logs were collected during the month of April 2012. In this month, the "Edge" contained 951 compute nodes and executed 120,639 jobs. In our data set, most of the jobs are short, and 58.1% of the jobs are shorter than 10 minutes. The cumulative distribution of job lengths is shown in Figure 2 (up to 20000 seconds). As can be seen in the figure, more than 80% of the jobs finish within 20,000 seconds.

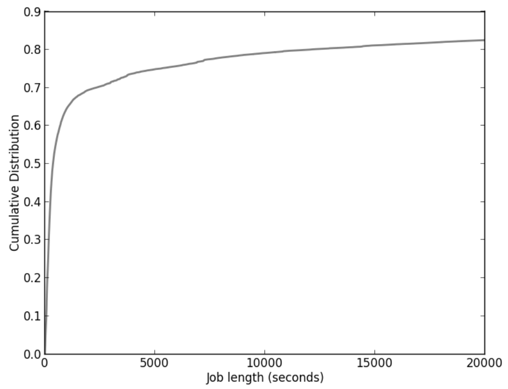


Fig. 2. Cumulative distribution of job execution times less than 20000 seconds.

The system logs in this study are contained in the /var/log/messages file on the compute nodes. This plain-text file is generated by the POSIX-standard "syslog" protocol [6], which allows *any* program to submit messages. Each line in the syslog file has the following format: `<timestamp>` `<hostname>` `<program>` `<text>`, where `<program>`

is a self-identified program name, e.g. kernel, pbs_mom, or dhclient, and `<text>` is a free-form string. Based on PBS batch job scheduler information, we slice the log messages to obtain per-job log message sets. In 94% of the jobs, there is no sharing of compute nodes among jobs. This is fairly common in supercomputing systems.

Figure 3 shows the distribution of the number of nodes occupied by each job through its lifetime. As can be seen in the figure, the overwhelming majority of jobs occupy a single node of the cluster.

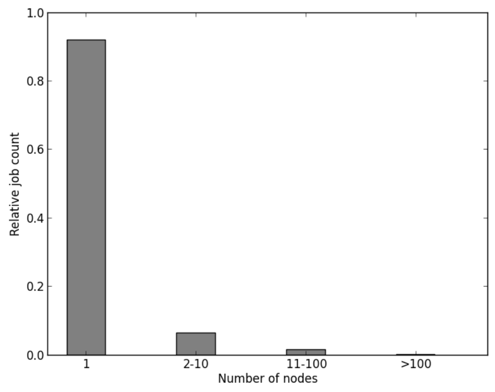


Fig. 3. The percentage of jobs and their occupying nodes in the category of 1, 2-10, 11-100 and larger than 100 nodes.

```
LOGS FOR JOB 2181416 BETWEEN 2012-04-26
23:54:48 TO 2012-04-27 06:57:47 USER 270130

NODE #1109/f15n14
Apr 27 06:57:01 f15n14 kernel: imklog 4.6.2,
    log source = /proc/kmsg started.
Apr 27 06:57:01 f15n14 rsyslogd: [origin
    software="rsyslogd" swVersion="4.6.2"
    x-pid="1173" x-info="http://www.rsyslog.
    com"] (re)start
Apr 27 06:57:02 f15n14 rpc.statd[1217]:
    Version 1.2.3 starting
Apr 27 06:57:04 f15n14 kdump: kexec: loaded
    kdump kernel
```

Fig. 4. Example log messages for a job

Figure 4 shows an example of part of a job's log message set. It contains a job ID (2181416 in this case), user ID (270130), start time (2012-04-26 23:54:48), finish time (2012-04-27 06:57:47), and a list of allocated nodes (in this case, single node, # 1109/f15n14).

C. Related Work

Log analysis has been successfully applied to debug systems, optimize system performance, detect security breaches, and diagnose computer system failures failures [10]. We focus on the last category as it is most pertinent to our research.

In systems specifically engineered for high reliability such as BlueGene, each log message is labeled with a severity level and source hardware component, so predicting failures and identifying the involved nodes can be performed by analyzing the frequencies of target message types or keywords [8], [11]. However, most commodity supercomputing clusters only have semi-structured logs composing free-form text, and extracting useful knowledge from free-form textual strings in the logs poses a major challenge.

Logsurfer [12] is a rule-based tool to sift the logs for problems. It has a very expressive syntax but requires experts such as system administrators to define the rules. Sisyphus [13] is another toolkit designed to classify and tag log messages based on an information-theoretic entropy method in [14]. The results can then be easily visualized to eyeball anomalies. However, it requires users to have a dictionary of rules for filtering.

Many studies focus on discovering "interesting" or "exceptional" patterns in the log text without a priori notion of "bad behavior". Statistical techniques [15], and program source code analysis [16] have been used to automate this task. However, these techniques have a high rate of false positives, and the final results still must be interpreted by human experts.

Our research avoids the previous difficulties by working on a well-defined problem with objective criteria for the answer quality, namely that of predicting job residual times based on log messages. Further, we use HMMs to avoid the problem of defining "normal" system behaviors and capture the dynamics of running a job.

Other work [17] has used HMMs to represent syslog bevaviors, but they assume a tagged system. Hidden Markov Models have also been applied in sensor networks to distinguish errors and malicious attacks [18]. To our knowledge, we are the first to use HMMs to residual time prediction in supercomputing systems without requiring any tags or annotations on the logs.

III. Approach

In this section, we describe how we use HMMs to predict residual times from log messages. Figure 5 shows the workflow. First, all log messages are *templatized* (see § III-A) and a templatized log message set is generated for each job. For simplicity, depending on the context, *when we refer to log messages, we mean log templates*. The second step is to choose important log messages based on sequence mining and frequency distribution analysis, and throw away those that are deemed unimportant or "noisy." In the third step, we partition the job-wise log message sets into training and testing sets. Then for each log message in the training data, we build a discrete HMM with an absorbing state, so each message has its own transition matrix and emission matrix. The job termination corresponds to the absorbing state, so in the final step, we calculate the *expected absorption time* to this terminal state, and this value is our job residual time prediction. This value is computed whenever an important log message is seen.

A. Generate Log Templates and Sequences

Before using the log messages to build the model, we need to turn them into structured forms. In the first step, the log

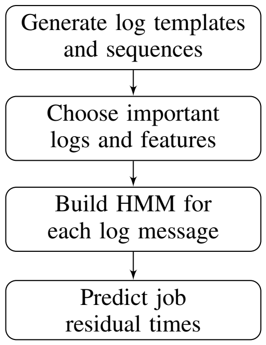

Generate log templates and sequences
Choose important logs and features
Build HMM for each log message
Predict job residual times

Fig. 5. The Prediction Workflow

messages are pruned into templates by extracting the words and removing punctuations, numbers and description information. For example, "puppet-agent[2210]: Finished catalog run in 7.83 seconds" becomes a template "puppet-agent Finished catalog run in seconds" after templatizing. The second step is to associate the timestamp, compute node ID, job ID, and user ID to each log template to obtain per-job log message sequences. An example of a per-job log template sequence is shown in Figure 6.

Time 0 :	Start
	...
Time t_1:	Important log template 1
	...
Time t_2:	Important log template 2
	...
	...
Time t_n:	Important log template n
	...
Time t_e:	End

Fig. 6. Important logs of a job

B. Important Log Templates and Features

To identify important log templates, we use a two-step process.

1) Identify messages that are more likely to occur towards the end of a job.
2) Identify message sequences consisting of messages identified in the first step.

The first step is to look for log templates that have a high frequency of occurrence near the end of a job, as such messages are crucial to job termination. Examples of such messages are listed in Table I. Note however, that not every log message that appears near the end of a job is important. For example, the log message "mpd: mpd ending' always appears at the end of jobs which use Intel MPI library. However, predicting job terminations based on this message may be futile, as it becomes too late to make any meaningful predictions. To choose important messages, we set a threshold of 0.6 for the message frequencies, and include any message that has higher frequency near the end of a job.

The second step is to identify frequent message sequences containing one or more of the message sequences identified in the first step. We consider message sequences as a message may be important if it is always succeeded by an important message, even if it itself does not appear near the end of a job. For example, the log templates "mpd: mpd starting" and "mpd: mpd ending" almost always appear at the beginning and the end of jobs respectively. In the dataset we have, 89.5% of "mpd: mpd starting" is followed by "mpd: mpd ending", and the "mpd: mpd starting" message allows more predictions at the early time of a job. More formally, we define $confidence$ as the conditional probability of a log template given another log template identified as important the first step. A high $confidence$ represents a strong correlation in the logs, implying that the second message is likely to be preceded by the first one. We tag such messages as important.

Example Log Template	Frequency
mpd: mpd ending mpdid inside cleanup	0.90178
puppet-agent: content change	0.908669
sssd: shutting down	0.904806
pbs_mom: req_cpyfile Unable to copy file	0.939497
puppet-agent: ensure changed stopped to running	1

TABLE I
LOG TEMPLATE EXAMPLES THAT APPEAR FREQUENTLY IN THE LAST 10% TIME OF JOBS.

C. Hidden Markov Models

The important log templates and sequences do not provide explicit residual time information, so we need a predictive model to quantify the residual times. We adopt HMMs for this purpose. The underlying assumptions are: (1) the actual system states are not directly visible and can be represent by the Markov chains, and (2) each log template corresponds to one or more system states, so each template has its own model.

Our HMM is shown in Figure 7. The hidden/unobservable state x_i models the time progress of a job. For the training dataset, each job's total run time is known a priori. We slice the entire run time of a job into 10 equal, non-overlapping intervals and assign 10 hidden states to the intervals, namely x_1 represents the first 10% of a job's run time, x_2 the second 10% of the run time, and so on. At the end, an extra state x_{11} is added as the absorbing state to denote the job termination.

The observation y_i is modeled by the frequency distribution of a log template. All log templates are classified into the following patterns: (1) "high frequency near the beginning", (2) "sparse and ubiquitous", (3) "dense and ubiquitous", (4) "high frequency near the end", and (5) "extremely high frequency near the end". Examples of each category are shown in Figure 8. The category of a log template acts as the observable state. Whenever a new log template appears and its category is different from the previous one, the observation is likely to be labeled as a feature of the current log template. We also add observation y_6 to correspond the absorption state x_{11}.

For each log template in the training dataset, we train for the model parameters $\lambda = (A, B, \pi)$ as mentioned in

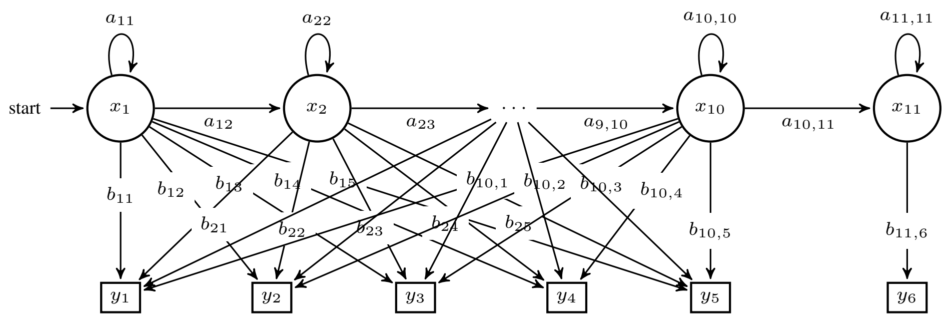


Fig. 7. HMM diagram. The x_i's correspond to the job progress and the y_i's map to the category of a log template.

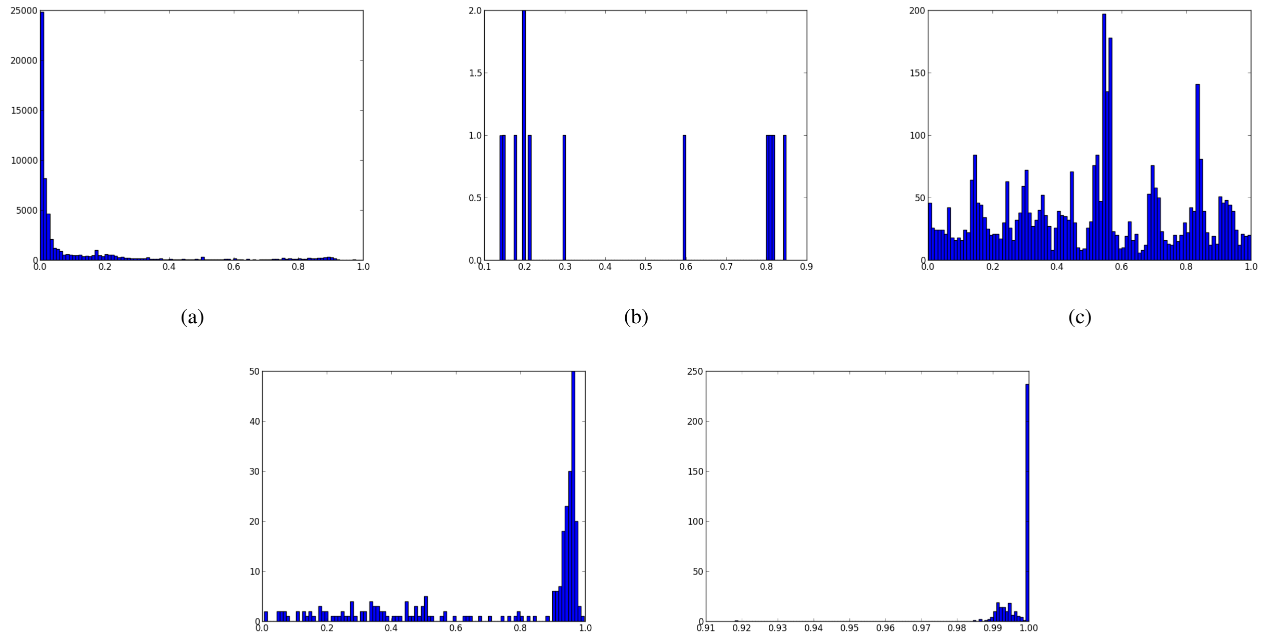


Fig. 8. Examples of categories of log templates: "high frequency near the beginning" (a), "sparse and ubiquitous" (b), "dense and ubiquitous" (c), "high frequency near the end" (d), and "extremely high frequency near the end" (e). The x-axis is the normalized appearance time of the log template in jobs and the y-axis is the frequency for all graphs.

§II-A. The training proceeds as follows. First, from the log template sequence of a job, we generate an output sequence composed of 1,2,3,4,5 and 6's, one number for each time step by rounding the timestamp of logs of a job to the nearest integer. If a time step does not have any log template, we use the prior step's log template. Thus, the state transition in the HMM takes place every time step until the absorption state is reached. The choice of the time step determines the speed of learning and its accuracy (Section IV).

A log sequence of a job always starts from the state x_1 and ends at x_{11}, and the initial probabilities π are fixed to be 1 for x_1 and 0 for the rest. With the output sequence as described, we compute the most likely hidden state transition sequence and the model parameters $\lambda = (A, B, \pi)$ using the Viterbi algorithm [9].

D. Predict Job Residual Time

After computing the model parameters for each log template, we predict job residual times as follows. Assume that we see the log template sequence of a job.

1) Start from the hidden state x_1, and compute its transition matrix A and emission matrix B.
2) When an important log message s appears, estimate its transition matrix A' and emission matrix B' and calculate the new model.
3) Calculate the posterior probabilities of being at each state at a specific time to decide the current hidden state given the observed sequence.
4) Calculate the expected time needed to transition to the absorbing state x_{11} of the HMM. This is the residual time.

IV. Experimental Evaluation

In this section we evaluate the proposed approach on the April 2012 log dataset and analyze the experiment results. We start by explaining our choice of the time step in the HMMs, followed by an evaluation of the accuracy of the overall method. We further refine the method using information from short jobs alone, and examine the resulting accuracy.

A. *Choosing the time step*

The rate of finding the optimal hidden state sequence for the observed sequence is proportional to the length of observed sequences in HMM. For a certain length of job, a smaller time step leads to a larger number of discrete time points, thus lengthening the observed sequence. Choosing a larger value of the time step thus trades off accuracy for learning time. To determine an optimal value of the time step, we randomly sample 5% of the log messages and use these to measure the prediction accuracy and computational time. These experiments were carried out on an Intel quad-core i7 machine with 8 GB RAM. The results are shown in Figure 9.

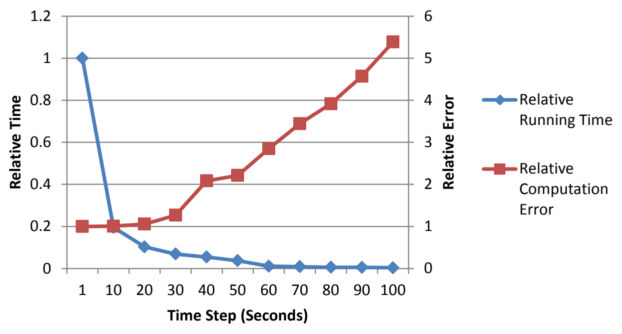


Fig. 9. The relative running time and relative error for different time step values (x-axis). The computation error and running time are both normalized (to fractions and percentages).

As can be seen in the Figure, there is a sharp drop in the running time when the time step is increased from 1 to 10 seconds. Further, increasing the time step beyond this point does not significantly reduce the running time, but decreases the overall accuracy considerably. Therefore, we choose a time step of 10 seconds in our experiments.

B. *Overall HMM results*

We use cross-validation to verify the effectiveness of our HMM approach. Each job and its log template set is viewed as a single sample. A round of cross-validation partitions the samples into training and testing sets, obtains the HMM parameters from the training set, and validates the HMM parameters against the testing set.

In particular, we proceed as follows. We first randomly partition the entire collection of jobs into ten equally-sized batches, numbered $1, 2, \ldots, 10$. For each $k = 1, 2, \ldots, 10$, we train for the HMM parameters on all but the k-th batch, and we test the resultant parameters against the k-th batch. Thus, the cross-validation is repeated ten times and the results are averaged across all batches.

Table II shows the actual states and the average distance from the predicted states of the HMM (with 95% confidence). As can be observed from the table, the error is lowest in the initial and final states, with the middle states having the highest error. This is intuitive, as in the initial states, the prediction is unlikely to forecast the end of a job, which would be correct, and in the final state, the prediction is likely to forecast the end of a job, which would also be correct. It is only in the middle states that making a prediction is problematic, as one does not have sufficient information to forecast when the job would end, and the predicted states tend to gravitate toward either the first or the final state.

State	Error	State	Error
1	0.37349 ± 0.067881	6	3.949961 ± 0.281477
2	1.42278 ± 0.134552	7	3.503469 ± 0.289571
3	2.48941 ± 0.269069	8	3.776115 ± 0.348699
4	3.46872 ± 0.272842	9	2.543330 ± 0.353124
5	3.91575 ± 0.070475	10	0.877631 ± 0.046800

TABLE II
ACTUAL STATES AND THE MEAN DISTANCE FROM THE PREDICTED STATES WITH 95% CONFIDENCE.

We also calculate the error between predicted and actual residual times. The histogram of the error is shown in Figure 10. As can be seen in the figure, 91.28% of the jobs have prediction error of less than 5000 seconds, and more encouragingly, 74.43% of the job have prediction errors of less than 200 seconds. In our data set, 70% of the jobs finish in 2279 seconds, and 90% of the jobs last less than 60069 seconds. Further, the average running time of a job is 17315 seconds. Therefore, an error of less than 200 seconds corresponds to a 10% error for 70% of the jobs, and is less than 2% of the average running time of a job.

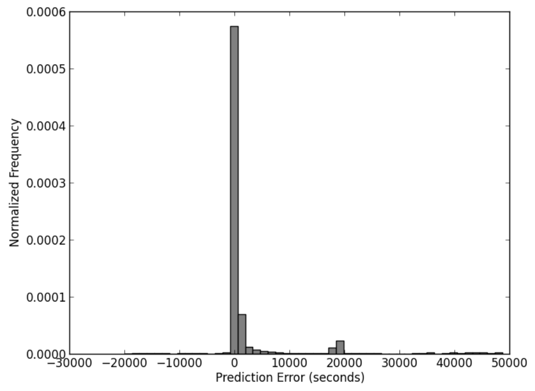
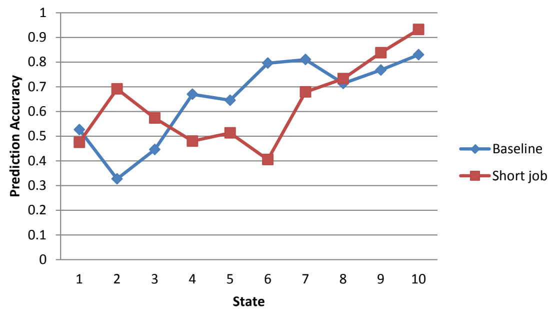


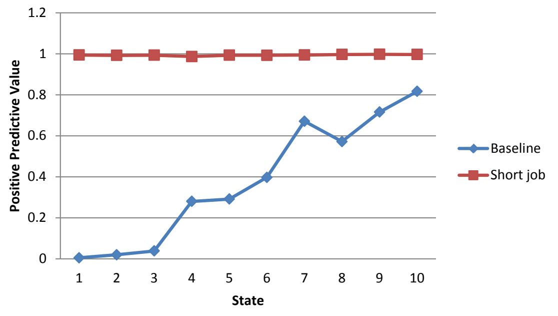
Fig. 10. The distribution of prediction errors. The x-axis is the error between predicted and actual residual times. The y-axis is the normalized frequency in the histogram.

C. Predicting Job Termination Within a Short Period

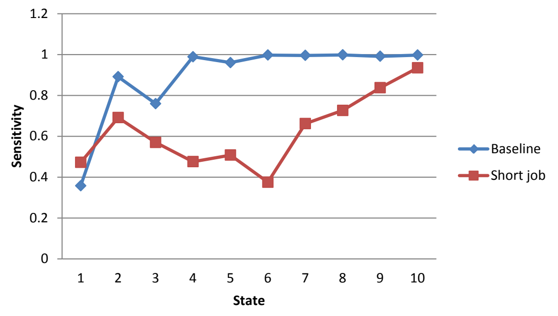
For many practical purposes, such as checkpointing a job just prior to job termination, it is not worthwhile predicting the precise residual time if a job still has several days to run. Therefore, we would like to predict whether a given job will finish in the next 10 minutes (or not).



(a)



(b)



(c)

Fig. 11. Predicting job termination within a short period using full dataset and short-job set. (a) The general predictive accuracy, (b) The positive predictive value, and (c) the sensitivity. The x-axis is the number of the state.

One challenge of predicting whether a job will end within the next 10 minutes is that jobs of different lengths contribute to variations in the accuracy of the model. The majority of the jobs are less than one hour long. Therefore, we reduce the variations by using only the short jobs as training dataset.

In this experiment, we used two datasets to build the HMM: the "baseline model" is the same as the original one and serves as the baseline, and the "short job model" is trained by jobs less than one hour long. Based on the prediction outcome, there are four possibilities:

1) True positive n_1: the job is predicted to end in 10 minutes and actually ends in 10 minutes
2) False negative n_2: the job is predicted to end in 10 minutes but does not end in 10 minutes
3) False positive n_3: the job is predicted not to end in 10 minutes and actually ends in 10 minutes
4) True negative n_4: the job is predicted not to end in 10 minutes and does not end in 10 minutes

where n_i denotes the number of predictions in the i^{th} category. We define the general predictive value $general\ accuracy$, the positive predictive value $positive\ accuracy$, and the sensitivity value $sensitivity$ as below:

$$general\ accuracy = \frac{n_1 + n_4}{n_1 + n_2 + n_3 + n_4}$$

$$positive\ accuracy = \frac{n_1}{n_1 + n_3}$$

$$sensitivity = \frac{n_1}{n_1 + n_2}$$

The general predictive value is the proportion of correct predictions in the entire dataset, the positive predictive value is the proportion of true positives in the positive test results, and the sensitivity is the proportion of true positives in all predicted positives.

Figure 11 shows the relative strengths of the two models for $general\ accuracy$, $positive\ accuracy$ and $sensitivity$ in different states, respectively. In Figure 11(a), the two models exhibit similar trends. At the beginning of a job, there is usually insufficient information to make accurate prediction of residual time. As the job progresses, more logs appear and important log templates help improve the accuracy of the prediction. Therefore, the $general\ accuracy$ increases with the increment of state number. Near the final state, the average accuracies of the baseline model and short job model are around 83% and 93% respetively. Thus, the short-job model improves the prediction accuracy by about 10% overall.

With regard to the positive predictive value, the short job model overwhelmingly outperforms the baseline model, as shown in Figure 11(b). The $positive\ accuracy$ of the short job model approaches 99%, compared to 80% for the baseline model. In other words, 99% of the jobs that the short job model predicts will end within the next 10 minutes, actually do so. This is because our dataset consists of a large amount of short jobs, e.g., less than 15 minutes. At the beginning of the jobs, the baseline job estimates that most of the jobs would not end in the next 10 minutes, but this contradicts the reality. Therefore, not surprisingly, the baseline HMM which is trained using the full dataset has poor predictability compared to the short jobs model.

The sensitivity is depicted in Figure 11(c). The baseline model has a $sensitivity$ near 1 during most of the states, which makes it vulnerable to the false negatives in the prediction. In contrast, the short jobs model has much lower sensitivity for most states, except near the end of the job, where its sensitivity matches that of the baseline model.

V. Conclusion and Future Work

In this paper, we address the problem of predicting job completion times in supercomputing clusters using Hidden Markov Models coupled with frequency-based important log message analysis. We model a job's time progress as hidden states of the HMM and the important log messages as the observed sequence. Our HMM approach can predict 75% of jobs within 200 seconds of error. If we train the HMM using short jobs, which are dominant in our job collection, predicting job termination in 10 minutes has a 1% false positive rate, and an overall accuracy of 93%.

In the future we will explore methods to tune our HMM approach for higher accuracy. For example, we would like to develop an on-line prediction system which continuously trains and updates itself using the latest job data. Further, we want to automatically mine and identify important log subsequences and patterns (instead of individual log messages) and build HMMs based on them. Finally, the HMM proposed in this paper has a fixed structure, e.g. the number of hidden and observable states, and we will devise mechanisms to better determine the structure of the HMM.

Acknowledgements: This work was funded in part by a Discovery Grant from the National Science and Engineering Research Council of Canada (NSERC), and a gift from Nokia Corporation. The log message data is provided by the Center for Computational Research at the State University of New York (SUNY) Buffalo. We thank the anonymous reviewers of the RSDA 2013 workshop for their insightful comments.

References

[1] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo, "Performance implications of failures in large-scale cluster scheduling," in *International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2004.

[2] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel job scheduling – a status report," in *International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2004.

[3] G. Bronevetsky, R. Fernandes, D. Marques, K. Pingali, and P. Stodghill, "Recent advances in checkpoint/recovery systems," in *IEEE Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[4] E. Elnozahy and J. Plank, "Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 97 – 108, 2004.

[5] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into HPC systems," in *IEEE/ACM Conference on High Performance Computing Networking, Storage and Analysis (SC)*, 2012.

[6] "http://www.infodrom.org/projects/sysklogd/."

[7] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *International Conference on Dependable Systems and Networks (DSN)*, 2007, pp. 575 – 584.

[8] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, "BlueGene/L failure analysis and prediction models," in *International Conference on Dependable Systems and Networks (DSN)*, 2006, pp. 425 – 434.

[9] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257 – 286, 1989.

[10] A. Oliner, G. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Communications of the ACM*, vol. 55, no. 2, pp. 55 – 61, 2012.

[11] Y. Zhang and A. Sivasubramaniam, "Failure prediction in IBM BlueGene/L event logs," in *IEEE Parallel and Distributed Processing Symposium (IPDPS)*, 2008.

[12] J. Prewett, "Analyzing cluster log files using Logsurfer," in *The 4th Annual Conference on Linux Clusters*, 2003.

[13] J. Stearley and A. Oliner, "Bad words: Finding faults in Spirit's syslogs," in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2008.

[14] A. Oliner, A. Aiken, and J. Stearley, "Alert detection in system logs," in *IEEE International Conference on Data Mining*, 2008, pp. 959 – 964.

[15] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset, "Analyzing system logs: A new view of what's important," in *Workshop on Computer Systems with Machine Learning (SysML)*, 2007.

[16] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Experience mining Google's production console logs," in *Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML)*, 2010.

[17] K. Yamanishi and Y. Maruyama, "Dynamic syslog mining for network failure monitoring," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, ser. KDD '05. New York, NY, USA: ACM, 2005, pp. 499–508. [Online]. Available: http://doi.acm.org/10.1145/1081870.1081927

[18] C. Basile, M. Gupta, Z. Kalbarczyk, and R. K. Iyer, "An approach for detecting and distinguishing errors versus attacks in sensor networks," in *Proceedings of the International Conference on Dependable Systems and Networks*, ser. DSN '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 473–484. [Online]. Available: http://dx.doi.org/10.1109/DSN.2006.11

[19] M. J. Zaki, "Sequences mining in categorical domains: Incorporating constraints," in *9th ACM International Conference on Information and Knowledge Management*, Nov 2000.