Fault prediction under the microscope: A closer look into HPC systems

Ana Gainaru
Computer Science Department
UIUC, Urbana, IL, USA
againaru@illinois.edu

Franck Cappello
INRIA
France
fci@lri.fr

Marc Snir
MCS
ANL, IL, USA
snir@mcs.anl.gov

William Kramer
NCSA
UIUC, Urbana, IL, USA
wkramer@ncsa.illinois.edu

***Abstract*—A large percentage of computing capacity in today's large high-performance computing systems is wasted because of failures. Consequently current research is focusing on providing fault tolerance strategies that aim to minimize fault's effects on applications. By far the most popular technique is the checkpoint-restart strategy. A complement to this classical approach is failure avoidance, by which the occurrence of a fault is predicted and preventive measures are taken. This requires a reliable prediction system to anticipate failures and their locations. Thus far, research in this field has used ideal predictors that were not implemented in real HPC systems.**

In this paper, we merge signal analysis concepts with data mining techniques to extend the ELSA (Event Log Signal Analyzer) toolkit and offer an adaptive and more efficient prediction module. Our goal is to provide models that characterize the normal behavior of a system and the way faults affect it. Being able to detect deviations from normality quickly is the foundation of accurate fault prediction. However, this is challenging because component failure dynamics are heterogeneous in space and time. To this end, a large part of the paper is focused on a detailed analysis of the prediction method, by applying it to two large-scale systems and by investigating the characteristics and bottlenecks of each step of the prediction process. Furthermore, we analyze the prediction's precision and recall impact on current checkpointing strategies and highlight future improvements and directions for research in this field.

***Index Terms*—fault tolerance; large-scale HPC systems; signal analysis; fault detection.**

I. Introduction

Current HPC systems waste a significant fraction of their capacity because of failures. The problem is expected to worsen in the coming years. Therefore, the problem of resilience at large scale has attracted significant research. Numerous fault tolerance strategies have been proposed but by far the most popular is the checkpoint-restart technique [16]. One way to reduce the overhead induced by these strategies is by combining them with failure avoidance methods [34]. Failure avoidance is based on a model that predicts fault occurrences ahead of time and allows preventive measures to be taken, such as task migration or checkpointing the application.

The analysis of event logs provides valuable information about the normal behavior of the system and how failures affect it, and it allows identifying system bottlenecks [20], [21]. Recent observations show different faults generate very different syndromes of logged events.

For example, a network file system error usually generates a burst of notifications in the log; however, when a node card fails, the event is usually represented by a lack of messages in the log [4].

Considerable research has been done on analyzing log files for prediction purposes, none of which takes into consideration the different characterization of failures [11], [29], [6], [18], [10]. Furthermore, most of the studies do not consider the failure location in their analysis, making reactive methods less effective. For checkpointing strategies, prediction with location information will allow the system to checkpoint data only on the failed components. For migration, only the tasks on failure prone components should be migrated. As we will show in our experiment section, these methods have limitations that can affect the overall performance of the prediction.

Moreover, it is important to be able to capture the behavior of each event type and understand the characteristics that change over time and how errors affect such behavior. Systems experience software upgrades, configuration changes, and even installation of new components during their lifetime [7], [21]. These make it difficult for the algorithms to learn patterns since the system will experience phase shifts in behavior.

In this paper we combine signal-processing concepts and data-mining techniques in the implementation of an analysis module for large-scale systems. Signal analysis allows us to characterize the behavior of events affecting the system, highlighting the differences between failures. Data mining is a powerful technology that efficiently extracts patterns in high-dimensionality sets and can provide accurate correlations between defined behaviors. We used the advantages of both methods by offering a hybrid approach for predicting failures in HPC systems. We also implemented a propagation module that extends the prediction with location information so that the results could be applied on current fault tolerance protocols.

Another important contribution of our paper is an in-depth analysis of our results compared to results obtained only from data-mining or only from signal analysis modules. We investigate in detail what error types are easier to predict, the limitations in correlating events or predicting faults, and the impact of having location information in the prediction step. To the best of our knowledge this paper is the first to combine signal analysis with data mining. We demonstrate the effectiveness of using such a model by implementing the prediction system and studying its impact on checkpointing strategies. Our prediction system takes into consideration the location of the error-prone component and the online analysis time.

The rest of the paper is organized as follows. Section 2 gives an overview of the related work, highlighting the advantages of our method. Section 3 presents the methodology behind the hybrid approach, giving details regarding each of the algorithms used. The next three sections give a detailed analysis of the correlation, location and prediction results respectively. We conclude with a summary of our work and a discussion of future research directions.

II. Related work

In this section we will give a brief overview of solutions proposed over the past few years and discuss their limitations. Also, we look at current checkpointing strategies and discuss their potential benefit in the presence of a prediction module.

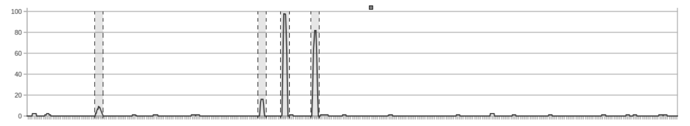
Researchers have used for failure prediction association rule based methods [5], decision trees, or Bayesian networks [32]. All these methods examine correlations between previous events and fatal events in order to predict potential failures in the future; see, e.g., [10], [11]: Both studies extract rules for a fixed time window and generate association rules between fatal and nonfatal events that leave a prediction window big enough so that proactive measures can be taken. In [13], the authors use a similar algorithm with the difference that the rule extraction can measure correlations of events that may happen in an interleaved way. In our earlier work [12], we used a method similar to the previous ones, but we adapted the prediction window according to the each event type. A different approach is given in [8] and [9], where the authors investigate parameter correspondence between different application log messages for extracting dependencies among different system components.

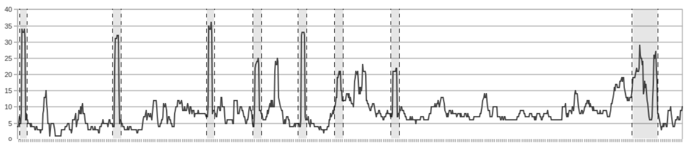
In [31], the authors introduce the concept of dynamic metalearning where the prediction engine switches between different methods depending on different rules.

Another approach for analyzing the logs is given in [14], where the authors investigate both usage and failure logs. For each failure instance, past and future failure information is accumulated; and based on these features, different decision tree classifiers are used in order to predict failures within a fixed time window.

None of these methods provide location information. Thus, it is impossible to use them in order to migrate processes that are likely to fail [30] or for performing a local checkpoint only on those failure-prone components, thereby avoiding time-consuming, system-wide checkpointing. [29] deals with predicting the location of a failure. However it considers that all errors share the same behavior. In fact, none of the presented papers make a difference between different events; rather, they analyze all events in the same way.

Another approach to address fault location is to develop fault propagation models, such as causality graphs or dependency graphs [33]. However, these require a priori knowledge about the system structure and dependencies among different components, information that is hard to obtain and maintain at large scales.


(a) L3 major internal error


(b) instruction cache parity error corrected

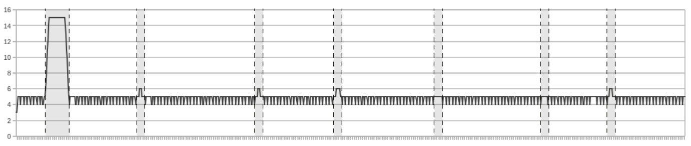

(c) Controlling BG/L rows

Fig. 1. Signals generated by HPC systems and the outliers in each

In this paper we propose a novel way of analyzing log files, by using signal analysis to characterize events and data-mining algorithms to find patterns between them regardless of their behavior. This approach gives a better understanding of the behavior of the entire system and provides a more accurate prediction.

The expectations for future exascale systems have accelerated the research for efficient checkpointing protocols. [25] proposes a low-overhead, high-frequency multi-level checkpoint scheme. Similarly, [28] proposes a multi-level checkpoint and a probabilistic Markov model that describes the performance of such a multi-level scheme. [26] uses a fault-tolerant protocol based on double in-memory checkpoint/restart and the idea of processor virtualization and migratable objects. For shared-memory applications, [27] provides a hardware-based coordinated local checkpointing for scalable coherence. The time to checkpoint differs significantly between different strategies; however some studies show that medium memory-intensive applications could be saved in around one minute [25].

Our prediction method complements this work by providing predictions that leave enough time for checkpointing and task migration. Consequently, our method has the potential of decreasing the overhead induced by these methods by reducing, for example, the checkpoint frequency.

III. Hybrid approach

In previous work [4], we introduced signal analysis concepts in the context of log file analysis. We discovered that events generated by systems are characterized by three types of signals: periodic, noise. and silent. Figure 1 presents the three types and the possible cause for each type.

We observed that a fault trigger in the system does not have a consistent representation in the logs. For example, a memory failure will cause the faulty module to generate a large number of messages. Conversely, in case of a node crash, the error will be characterized by a lack of notifications. Data-mining algorithms in general assume that faults manifest themselves in the same way and in consequence fail to handle more than one type behaviors.

For example, even though silent signals represent the majority of event types, data-mining algorithms fail to extract the correlation between them and other types of signals. This situation affects fault prediction in both the total number of faults seen by the method and in the time delay offered between the prediction and the actual occurrence of the fault.

Signal analysis methods can handle all three signal types and thus provide a larger set of correlations that can be used for prediction. However, data mining algorithms are more suited for characterizing correlations between different high dimensionality sets than the cross correlation function offered by signal analysis. Data mining is a powerful technology that converts raw data into an understandable and actionable form, which can then be used to predict future trends or provide meaning to historical events.

Additionally, outlier detection has a rich research history in incorporating both statistical and data mining methods for different types of datasets. Moreover, they are able to implicitly adapt to changes in the dataset and to apply threshold based distance measures separating outliers from the bulk of good observations. In this paper, we combine the advantages of both methods in order to offer a hybrid approach capable of characterizing different behaviors given by events generated by an HPC system and providing an adaptive forecasting method by using the latest data-mining techniques.

In the following sections we present the methodology used for preprocessing the log files and extracting the signals. We then introduce the novel hybrid method that combines signal analysis concepts with data mining techniques for outlier detection and correlation extraction. An overview of the methodology is presented in Figure 2.

A. Preprocessing

Log files generated by a large HPC system contain millions of message lines, making their manual analysis impossible. Moreover, the format in which messages are generated is not structured and differs between different system and sometimes even between different components of the same machine. To bring structure to our analysis, we extract the description of all generated messages. These descriptions represent the characterization of different events used by the system. Also, as software changes with versions, bug fixes, or driver updates, these descriptions are modified to reflect the system's output at all times.

We performed an initial pass over the logs files in order to identify frequently occurring messages with similar syntactic patterns. Specifically, we use the Hierarchical Event Log Organizer [15] on the raw logs, resulting in a list of message

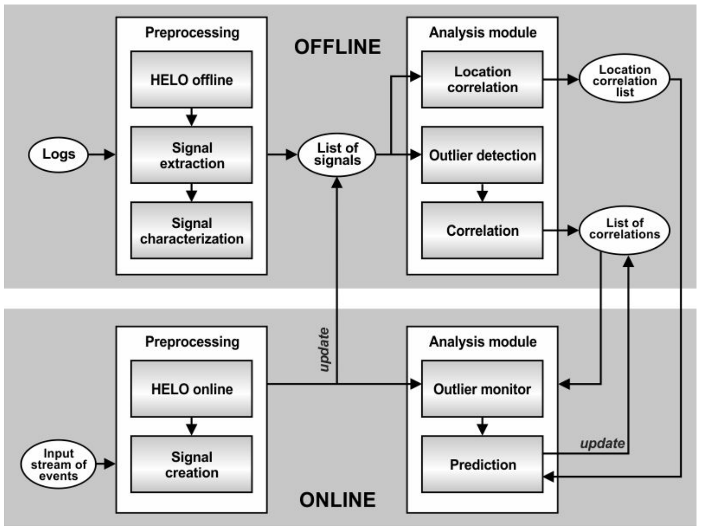


Fig. 2. Methodology overview of the hybrid approach

templates. These templates represent regular expressions that describe a set of syntactically related messages and define different events in a system. In the on-line phase, we use HELO on-line to keep the set of templates updated and relevant to the output of the system.

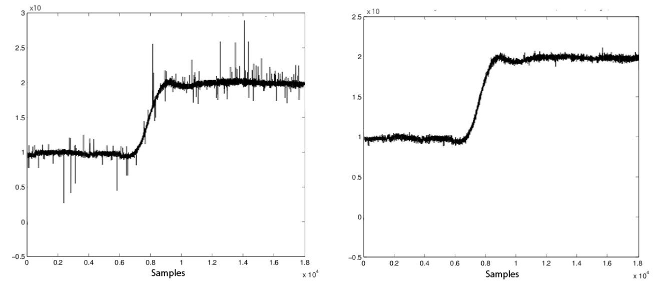
For the rest of the paper, we analyze the generated events separately by extracting a signal for each of them and characterizing their behavior and the correlations between them. Figure 1 presents one template or event type for each type of signals. First, we extract the signal for each event type by sampling the number of event occurrences for every time unit. Afterwards, we use wavelets and filtering to characterize the normal behavior for each of them. In our experiments, we use a sampling rate of 10 seconds for all signals. More details about this step can be found in [4].

In the on-line phase, the signal creation module simply concatenates the existing signals with the information received from the input stream of events. For optimization purposes, we keep only the last two months in the on-line module since execution time is an important factor in this phase. The outline monitor and the prediction system are applied on this trimmed and updated set of signals.

B. Analysis Modules

1) Outlier detection: All analysis modules are novel hybrid modules that combine data-mining techniques with the previously extracted set of signals and their characterization. Since the off-line phase is not run in real time and the execution time is not constrained, we do not optimize this step. For outlier detection in the on-line phase, we use as input the adapted set of signals and apply a simple data cleaning method for identifying the erroneous data points.

We implement this step as a filtering signal analysis module so that is can be easily inserted between signal analysis modules. The transformation was intuitive, since the data mining algorithm is based on a causal moving data window



(a) Original data (b) Signal after filtering

Fig. 3. On-line outlier detection

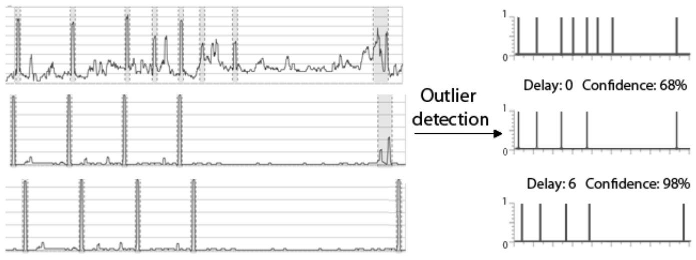


Fig. 4. Correlation example between three signals

that is appropriate to realtime applications: the observed data point y_k is compared with the median y_k^m of past data points, both the erroneous and the replaced ones. If the distance between these points is large relative to a threshold based on the normal behavior of the system, y_k is declared an outlier and a replacement with a more reasonable value y_k^c is proposed. Figure 3 presents a synthetic noise signal in its original form and after applying the on-line outlier detection with replacement for the erroneous data points.

For a window of N points, the analyzed list of points for the current y_k is

$$V_k = \{y_{k-N}^c, ..., y_{k-1}^c, y_{k-N}, y_{k-N+1}, ..., y_k\},$$

out of which the median y_k^m is extracted. For our experiments we use an N value of two months.

We use predefined thresholds for each signal, specified automatically in the preprocessing step based on knowledge about the normal behavior of the event type and how this was affected by outlier in the offline phase.

The replacement strategy decreases the influence of severe outliers on signals by saving both the initial value and one that is more consistent with the rest of the data set. At the same time it minimizes the effects of a large number of faults hitting the same signal for a larger period of time.

Having a low execution time is a requirement for the on-line modules. The on-the-fly filter makes the process faster than what is proposed in [4]. We will show in the experiment part the number of faults missed because the outlier detection and prediction took too long to notify the event with both the methods. At the same time, this data mining algorithm adapts to changes in the underlying signal.

C. Signal Correlation

Gradual itemset mining is used in the data-mining community for extracting patterns of the form "the more/less X_1, .. , the more/less X_n". The goal of the algorithm is to discover frequent covariations between different attributes. This method has the advantage of extracting multiple event correlations instead of only pairs, like the output of the signal cross-correlation function.

We use the sequential GRITE algorithm presented in [2], adapting it to work with our signals. Since the purpose of this paper is to predict faults, we are correlating signals based on the occurrences of outliers in them. For this end, we filter out the normal behavior and leave only the outliers.

To simplify the correlation process, we replace each point in the signal with 0 in case of normal behavior and 1 for outliers. This allows us to represent signals as attributes that can be handled by the gradual itemset mining algorithms. This transformation is illustrated in figure 4.

The sequential algorithm relies on a tree-based exploration, where each level is built by using information from the previous level. In its original form, the first level of the tree is initialized with all attributes. In our case, however, the initial level is composed of the 2-pair correlations obtained with the signal cross-correlation function. Gradual itemset mining is a computationally expensive data-mining algorithm, so sequential methods cannot yet scale to large datasets. By merging it with a fast signal analysis module we were able to guide the extraction process toward the final result, thereby reducing the complexity of the original data-mining algorithm. Recent research on gradual itemset mining has focused on parallel methods that are able to use multi-core architectures [3]. We plan to investigate the use of such methods on-line in order to adapt correlations to changes in the system.

Itemsets from the L level are computed by combining frequent itemsets siblings from the L-1 level by using a procedure for joining two itemsets into a larger one. Candidates that are more frequent than a predefined threshold are retained in level L and are used in the next level.

In its usual form, gradual itemset mining algorithms look for patterns that take place at the same time in a subset of attributes. For our purposes, we are interested in associating signals that have a fixed delay one from another. For example, if one event type usually occurs T time units after another event type, these two signals will be shifted with T time units one from the other. (For example, in Figure 4 the last two signals have a time delay of one minute.) The correlation module must be able to capture this scenario.

We modify the initial algorithm to check different delays between signals by shifting one of the signals with the corresponding delay and applying the gradual itemset mining algorithm. To optimize the process, we choose a small time window for the delay values based on the results given by the initial cross-correlation function.

The general gradual itemset mining algorithm uses a comparison operator in $\{\geq, \leq\}$. In our case, we care only about the decreasing patterns (if an outlier occurs in S_1, we want to find all other S_i signals where an outlier occurs with a fixed delay). We change the algorithm to only search for the $\geq$ operator.

For a better understanding of our hybrid approach we will present an example in the next paragraph that goes through all the steps used by the method. Given a table set of signals S, a gradual item is a pair (S_i, θ_i) where S_i is an attribute in S and θ_i represents a delay in the signal. A gradual itemset $G = \{(S_1, \theta_1), ..., (S_k, \theta_k)\}$ is a set of gradual items of cardinality greater than or equal to 2.

In the example illustrated in Figure 4, the initial set of gradual itemsets, which is given by the cross-correlation function between all combinations of signals, is $S_{G_{init}} = \{\{(S_1, 0), (S_2, \theta_{12})\}, \{(S_1, 0), (S_3, \theta_{13})\}, \{(S_2, 0), (S_3, \theta_{23})\}\}$.

The join function used in GRITE will return the merge between the sets in $S_{G_{init}}$ and create $S_{G_1} = \{\{(S_1, 0), (S_2, \theta_{12}), (S_3, \theta_{13})\}, \{(S_1, 0), (S_2, \theta_{12}), (S_3, \theta_{12} + \theta_{23})\}, ...\}$, with different delays. If all delays are consistent (for example if $\theta_{13} = \theta_{12} + \theta_{23}$) S_{G_1} will have only one element. The testing part is left almost unchanged from the gradual itemset mining, with the difference that we use only one operator.

We use the Mann Whitney test [22] to decide when a correlation is statistically significant. The output of this module is represented by a list of n-tuples correlations $Corr = \{G_1, .., G_m\}$ where each G_i is a gradual itemset of different sizes. We provide an in-depth analysis of the extracted correlation list in the experiments sections.

D. Location Correlation

Large-scale systems contain a large number of nodes that are organized in an hierarchy. For example for the BlueGene systems, nodes are gathered into midplanes and multiple midplanes form a rack. When analyzing different errors that might affect a HPC system we investigated the propagation behavior of each of them. Our observation show that some errors influence multiple nodes, depending on their location in the machine [20].

After a closer analysis, we observed that the propagation path for different error types follows closely the way components are connected in the system. For example, if a fan breaks, all nodes sharing the same rack will be affected. However, in general, the topology of a system is not available. This forces failure prediction algorithms to rely on heuristics for tracking the locations of the failures effects in the system. For a better understanding of the behavior of different event types, we analyzed the logs generated by Blue Gene/L and Mercury systems. Blue Gene is one of the few systems that logs information about the location of events, making it easier to understand the propagation behavior of correlated events.

The heuristic used to extract location correlations is based on the offline correlation chains extracted in a previous step. We parse the logs and monitori each occurrence of a correlation $G_i = \{(S_1, \theta_1), ..., (S_k, \theta_k)\}$. Based on it we extract the list of possible locations for each chain $Loc_i = \{(L_{11}, .., L_{1k_1}), ..., (L_{m1}, .., L_{mk_m})\}$, where $(L_{11}, .., L_{1k_1})$ is a list of unique locations where events in the chain have occurred and m is the number of occurrences for the corresponding sequence of events. In case of a correlation that does not propagate events from one node to another, the list of locations will be composed of only one element for each occurrence: $Loc_i = \{(L_1), (L_2), ..., (L_m)\}$.

IV. Dissecting Event Correlation

Our experiments are made mostly on the Blue Gene/L machine. The logs are available on-line at [24], and more information about the system can be found in [1]. The Blue Gene machine is organized into 32 midplanes and 64 racks. Locations are denoted by codes that represent different hardware components, such as compute nodes (R00-M0-N0-C:J02-U01), I/O nodes (R22-M0-N0-I:J18-U01), and node cards (R00-M0-N0). The location gives informations about the component's place in the architecture hierarchy as well. The logs we analyzed are from June 2005 to January 2006; they contain 207 different event types.

Most modules from our framework are platform independent and so are easy to adapt to run on different machines. To demonstrate this and to compare the results from different systems, we made additional experiments on the Mercury system. Mercury is a cluster at the National Center for Supercomputing Applications (NCSA) [23]. The generated logs are not available to the public because of privacy issues. The Mercury cluster started with 256 compute nodes; later 635 compute nodes with faster processors were added. We analyzed logs generated in 10 months of production from February 2006 to December 2006. Mercury logs contain 409 different event types.

Both systems were used in production with a variety of codes executing for the time frames analyzed in the logs. For example, two of the applications that ran on the Mercury systems are the NAMD application for biomolecular simulations and the CM1 code for atmospheric research. For our analysis purposes, our framework gets information only from the log files, hence the analysis does not change for memory intensive or communication intensive applications.

We used the first 3 months as training for the off-line phase and the rest of the logs for testing purposes.

A. Results

In this section we analyze the correlations we were able to extract with our method. First, we were interested understanding what type of patterns our method is able to extract in general. Table I presents a couple of examples returned by our method. We observed that the method detected not only sequences of events that led to a failure but also relations between informational messages. For example, multiline messages are identified by HELO as multiple event types. However, they have the same behavior so our correlation clusters them together.

Messages generated during the installation of a component or during a restart are another example of informational messages. Our tool characterizes their behavior as silent signals,

Memory error
correctable error detected in directory *
after 6 time units (one minute)
uncorrectable error detected in directory *   capture first directory correctable error address..0
after 1 time unit
DDR failing data registers: * *   number of correctable errors detected in L3 EDRAMs.*   parity error in read queue PLB.*
Node card failure
midplaneswitchcontroller performing bit sparing on * bit *
after 44 time units
linkcard power module * is not accessible
after 4 time unit
problem communicating with service card, ido chip: * java.io.ioexception: could not find ethernetswitch on port:address 1:136
after 6 time unit
prepareforservice is being done on this part * mcardsernum(*) * * mtype(*) by *
Multiline messages
general purpose registers: lr:* cr:* xer:* ctr:*
Component restart sequence
idoproxydb has been started: $name: d+ $ input parameters: -enableflush -loguserinfo db.properties bluegene1   ciodb has been restarted.   bglmaster has been started: ./bglmaster –consoleip 127.0.0.1 –consoleport 32035 –configfile bglmaster.init –autorestart y   mmcs_db_server has been started: ./mmcs_db_server –usedatabase bgl –dbproperties * –iolog /bgl/bluelight/logs/bgl –reconnect-blocks all n+

TABLE I
SEQUENCE OF CORRELATED EVENTS

since most of the time they do not appear in the log. Every time a restart occurs, these event types' occurrences are regarded as outliers. Our system correlate these signals with other event types and extract the complete restart sequence. However, these sequences do not help predicting failures since they do not affect an application's execution in any way.

We investigated what correlations that are not useful in the prediction phase. Doing so turned out to be a complex task since some messages might indicate harmless events in some contexts and indicate failures in others. Therefore, we only eliminated the obvious non-error sequences and analyzed the rest separately, as described in the prediction section of this paper.

We observed that only around 23% of sequences do not have any potential of predicting a problem in the system. We limited our analysis the Blue Gene/L machine because it offers a severity field that helped us in determining if a event type could be a failure in at least one context. We eliminated these sequences for the rest of the analysis. For the Blue Gene/L system this was done automatically by eliminating all sequences that contain only event types with INFO severity messages. For other systems, a system administrator would have to parse the correlation list and determine which ones can predict failures.

B. Analysis

We focus next on an in-depth analysis of the properties of the extracted correlations.

We investigated how many events are, on average, part of a correlation chain. We plot the distribution of the event types that compose a sequence in figure 5. The figure shows that, in general, the sequences contain a small number of event types; the average length of the chain is 4 for both systems. However, some correlations contain more event types, 20% of them containing more that 8 events.

Next, we analyzed the time delay between correlations found by our system. First, we analyzed simply the pair of initial correlations and then the complete sequences. We observed that 33.7% of the correlations have less than a 10 second delay between events, the majority (56%) having delays between 10 seconds and one minute and the rest having time delays of more than one minute. For both systems, only around 2.5% of the sequences have more than 10 minutes between events. For a better understanding of how this large percentage of correlations affects the final prediction, we analyzed the complete sequences as well.

We plot the time delay distribution between the first message indicating the beginning of a sequence and the last visible symptom. Figure 6 presents the results only for Blue Gene/L, because of space limitations, but Mercury has a similar distribution. We observe that the peak delay shifted to the right compared with the correlation delays. Only 12.8% of the sequences do not offer any prediction window larger than 10 seconds, 48.4% correlations offer between 10 seconds and one minute, and there is a significant percentage with a delay larger than one minute. Moreover, the correlation system is able to extract some sequences with hours time delay between the first symptom and the final failure message.

We also observe a relatively simple relation between the confidence of a sequence and the delay between the marginal events in the sequence. In general, for delays larger than 5 minutes, the larger the delay the lower the similarity degree between the signal and so the lower the confidence. Sequences

CIODB sequence
All happening at the same time
FAILURE ciodb exited abnormally due to signal: aborted FAILURE mmcs_server exited abnormally due to signal: * n+ SEVERE job * timed out. n+

Node card sequence
More than one hour difference between the first event and the last
WARNING endserviceaction is restarting the nodecards in midplane * as part of service action d+ SEVERE node card vpd check: * node in processor card slot * do not match. vpd ecid d+ found d+ SEVERE link card power module * is not accessible FAILURE no power module * found found on link card FAILURE temperature Over Limit on link card

TABLE II
SEQUENCES OF CORRELATED EVENTS WITH DIFFERENT TIME DELAYS

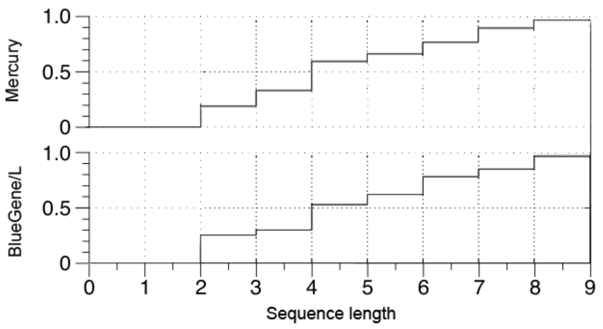


Fig. 5. Sequence size distribution

with a confidence of over 95% usually contain the correlations between events that are generated close in time. However, some node card failure sequences do have high confidence, yet offer more than a one-hour prediction window.

In the following section, we look more closely which failures have extreme time delays. In general, we observed that node card failures offer sequences with longer time delays. This situation should be reflected in a larger prediction window for these kind of errors, and as a consequence more time for fault avoidance strategies. For example, the node card failure presented in table I have around 9 minutes (the equivalent of 54 time units) between the first and the last event in the sequence. Other node cards examples show even one hour after the first symptoms occurs. The memory errors detected with our system, like the one presented in table I, usually offer, on average, a one-minute prediction window.

The Blue Gene/L system has a separate process, CIODB, that runs on service nodes and handles the job loading and starting. This process starts and monitors jobs and updates the job table as the job goes through the states of being loaded, started, and terminated. We observed that sequences or events related to CIODB usually have a very short time delay between them, the majority happening almost at the same time. Table II presents two extremes, a ciodb errors that gives no prediction window and a node card failure sequence that generates warnings with more than one hour before the actual failure events happen.

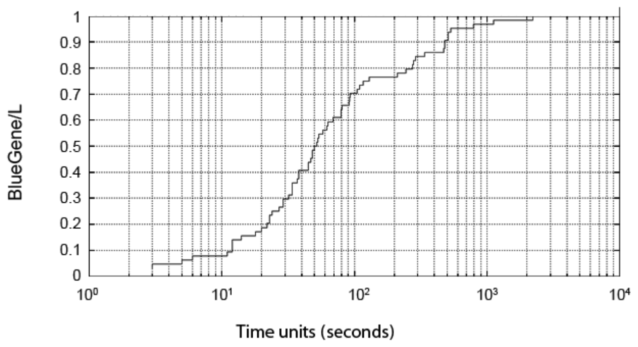


Fig. 6. Time delay distribution between events in sequences

In all our experiments, we used logs that offer less than 10 months of activity. For this reason, the correlation updating modules were not tested, since the changes in such a short time are not relevant to the whole lifetime of a system. We plan to make further experiments on larger periods of time.

V. DISSECTING EVENT PROPAGATION

In this section we investigate the propagation path taken by different extracted correlation chains. We wanted to know how many events propagate to different locations and how many locations are affected by one event.

We observed that, in general, sequences do not propagate to different locations and if they do propagate, they affect a small number of nodes. Only around 22% of the chains for Mercury and 25% for Blue Gene/L show any kind of propagation. Between 80% and 85% of the the sequences that show a propagation behavior affect less than 10 nodes. The rest, which represent less than 2% from the total number of correlations, influence a large number of nodes. An example of such a global failure is seen on the Mercury machine, when we investigated NFS (network file system) problems. The event "*rpc: bad tcp reclen d+ (non-terminal)*" indicates network file system unavailability to any requests for a machine. In applications using the network file system this could cause file operations to fail and the application to quit. Also, all nodes from which the application tries to access the network file system will be affected by this problem. This failure usually occurs nearly simultaneously on a large number of nodes.

To get a more realistic view of the behavior of sequences, we analyzed the initial pair of correlated events for the Blue Gene/L machine and broke down the propagation by racks, midplanes and nodes. Figure 7 shows that around 75% of correlations show no propagation at all and only around 2.16% extend outside of a midplane.

From our observation, in the Mercury system network failures can occur nearly simultaneously on multiple nodes. For example, the event "*ifup: could not get a valid interface name: -> skipped*" represents an unexpected node restart caused by unexpected hardware failure and propagates across different nodes. In general, errors in memory or processor caches do not show the same behavior. On the other hand, in the Blue Gene/L system we observed that some memory

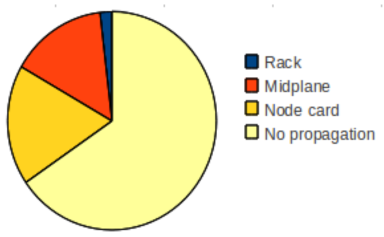


Fig. 7. Percentage of sequences propagating on different racks, midplanes and nodes

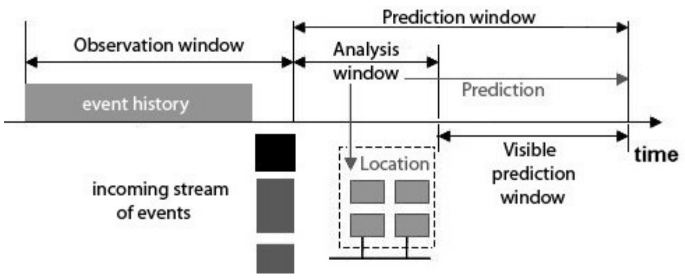


Fig. 8. Prediction time window

locations expend to different node cards in a midplane. For example, the sequence

*d+ ddr errors(s) detected and corrected on rank 0, symbol * bit * total of * ddr error(s) detected and corrected*

refers to a ddr memory error that was detected and corrected in a certain location and in most of the cases affects multiple nodes in the same midplane in a short period of time.

On the other hand, errors related to node cards do not propagate on multiple locations. For example, the sequence

can not get assembly information for node card
*linkCard power module * is not accessible*
*no power module * found found on link card*

gives information about a node card problem that is not fully functional. Events marked as "severe" and "failure" occur after around one hour; they report that the link card module is not accessible from the same midplane and that the link card is not found. The sequence is generated by the same node for all its occurrences in the log.

For 75% of correlations that do not propagate, the prediction system does not need to worry about finding the right location that will be affected by the failure. However, for the other 25% that propagate, a wrong prediction will lead to a decrease in both precision and recall. We analyzed this case a little further and observed that for most propagation sequences the initiating node (the one where the first symptom occurs) is included in the set of nodes affected by the failure. This observation leads us to believe that the recall of the prediction system will be more affected by the location predictor than its precision.

VI. Dissecting Prediction

Figure 8 shows an overview of the prediction process. The observation window is used for the outlier detection. The analysis time represent the overhead of our method in making a prediction: the execution time for detecting the outlier, triggering a correlation sequence, and finding the corresponding locations. The prediction window is the time delay until the predicted event will occur in the system. The prediction window starts right after the observation point but is visible only at the end of the analysis time.

In the next subsection we analyze the prediction based on the visible prediction window and then propose an analytical model for the impact of our results on checkpointing strategies. The metrics used for evaluating prediction performance are prediction and recall:

- Precision is the fraction of failure predictions that turn out to be correct.
- Recall is the fraction of failures that are predicted.

A. Analysis

In the on-line phase the analysis is composed of the outlier detection and the module that triggers the predictions after inspecting the correlation chains. We computed the execution time for different regimes: during the normal execution of the system and during the periods that put the most stress on the analysis, specifically periods with bursts of messages. If the incoming event type is already in an active correlation list, we do not investigate it further since it will not give us additional information.

The systems we analyzed generate, on average, 5 messages per second; message bursts generate around 100 messages per second. The analysis window is negligible in the first case and around 2.5 second in the second. The worst case seen for these systems was 8.43 seconds during an NFS failure on Mercury. Taking this analysis window into consideration we examined how many correlation chains are actually used for predicting failures and which failures are detected before they occur.

Our previous work showed 43% recall and 93% precision for the LANL system by using a purely signal analysis approach. At that point, however, we did not try to predict the location where the fault might occur. In this paper, we focus on both location and the prediction window. We compute the results only for the Blue Gene/L systems and guided our metrics based on the severity field offered by the system.

We analyzed the number of sequences found with our initial signal analysis approach, the data-mining algorithm described in [29], and the present hybrid method. Signal analysis gives a larger number of short-length sequences, thus making the analysis window longer. Also, the on-line outlier detection puts extra stress on the analysis making the analysis window exceed 30 seconds when the system experiences bursts. Because of our data-mining extraction of multi-event correlation we were able to keep only the most frequent subset making the on-line analysis work on a much smaller correlation set. On the other hand, the data-mining approach looses correlations between signals of different types, so even if the correlation set is much smaller than our hybrid method, the false negative count is higher.

Table III shows the precision and recall obtained with the three methods. The recall value for the signal analysis method is lower than in our previous findings. This can be explained

Prediction Method	Precision	Recall	Seq Used	Pred Failures
ELSA hybrid	91.2%	45.8%	62 (96.8%)	603
ELSA signal	88.1%	40.5%	117 (92.8%)	534
Data mining	91.9%	15.7%	39 (95.1%)	207

TABLE III
PERCENTAGE WASTE IMPROVEMENT IN CHECKPOINTING STRATEGIES

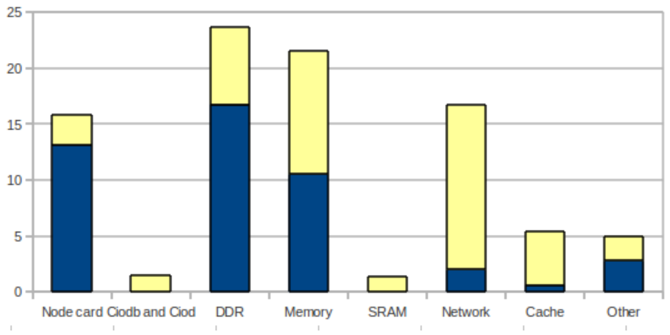


Fig. 9. Recall breakdown on different categories

by the location prediction, since now there is room for error in this part as well. What is interesting is that the precision value for the data-mining approach is higher than the other methods. The reason is that the sequences found by the data-mining method are mostly the ones that do not show a propagation behavior. When running our method without checking the location, we obtain a precision of around 94%. The results show that the hybrid method allows combining the precision given by the data-mining approach and the recall of the signal analysis method.

In a more detailed analysis, we break down the predicted events into different categories. The results are presented in Figure 9, where each bar represents how often a certain type of error appears in the log as a percentage reported to all errors in the system. The dark portion of every bar represents the correctly predicted cases out of the total occurrences. We observed that the node card errors were the type that our system detected with a high rate; more than 80% of the occurrences were predicted. This is explained in the high confidence sequences obtained for this type in the offline section. We plan to analyze the reason for such a low percentage in detecting network and cache failures.

The total number of error messages in the log represents 18% of everything recorded in the log. We observed after this analysis that even though the large majority of correlations are used at least once, there is a small set that is used frequently. More exactly, 3.12% of sequences are never used for prediction (the events occur only in the training set), and 23.4% are used in the majority of the cases.

We also analyzed the visible prediction window offered by the sequences used in this process and observed that around 85% of the prediction offer more than 10 seconds after the analysis window ended, out of which more than 50% offer more than one minute and around 6% more than 10 minutes.

This means that fault avoidance techniques that take a

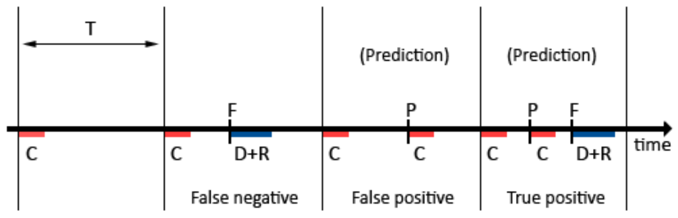


Fig. 10. Checkpoint-Restart mechanism

checkpoint or migrate a process in less than one minute could be applied on 42% of the total predicted failures on Blue Gene/L, respectively 20% of total failures. When using a fast checkpointing strategy, like the one from [25], the total number of failures for which avoidance techniques could be applied increases to 40%.

B. Impact on Checkpointing Strategies

In this section we derive an analytical model for the impact of prediction on adaptive checkpointing strategies, in order to highlight the benefit brought by our method.

If no failure prediction is available, then fault tolerance mechanisms must use periodic checkpointing and rollback recovery. We start from the model in [34], which computes the waste of a coordinated checkpointing strategy when no prediction is offered; we then integrate the impact of precision and recall on this model. Figure 10 presents the variables used in creating the model. T is the checkpoint interval, W is the percentage of wasted time and MTTF the mean time to failure for the application. Also, we assume that an application can be checkpointed locally at each node in C seconds, and that the checkpoint can be loaded back into memory in R seconds. Also, the downtime of a node and the time to restart the application on a different node or the same node in case of rejuvenation is assumed to be D seconds.

We assume that we are able to predict a fraction N of the failures with a precision of P, with N,P$\epsilon[0,1]$. We assume the failure distribution for the non-predicted failures remains exponential and that preventive actions are taken before the failure occurs for all predicted failures.

In case of no prediction, we start with the following model:

$$W = \frac{C}{T} + \frac{T}{2MTTF} + \frac{R+D}{MTTF}. \tag{1}$$

The formula accounts for the loss of C seconds every T seconds for taking checkpoints, the loss due to faults that occur every MTTF seconds and lose an average of $\frac{T}{2}$ time-steps each time and, in the last term, for the loss due to the recovery time that is taken for every failure.

The optimal checkpointing interval can be used to minimize waste; it is given by Young's formula:

$$T_{optimum} = \sqrt{2C \cdot MTTF}. \tag{2}$$

We now introduce the prediction model. First we assume having a recall of N and perfect precision. In this case the

MTTF of the unpredicted events will become:

$$MTTF_{new} = \frac{MTTF}{1-N} \tag{3}$$

For example if 25% of errors are predicted, the new MTTF is $\frac{4MTTF}{3}$. The rest of the failures are predicted events and have a mean time between them of $\frac{MTTF}{N}$ seconds.

By applying the new MTTF for the un-predicted failures from (3) to equation (2), the new optimal checkpoint interval becomes:

$$T_{optimum} = \sqrt{2C\frac{MTTF}{1-N}}. \tag{4}$$

The first two terms from equation (1) need to change to consider only the un-predicted failures since for all the others preventive actions will be taken. By adding the first two terms and incorporating the value for the checkpoint interval from equation (4), the minimum waste becomes:

$$W_{min}^{recall} = \sqrt{\frac{2C(1-N)}{MTTF}} + \frac{(R+D)}{MTTF}. \tag{5}$$

The last term from equation (1) will not change since for all failures, both predicted and un-predicted, the application needs to be restarted. Additional to the waste from (4), each time an error is predicted, the application will take a checkpoint and it will waste the time execution between this checkpoint is taken to the occurrence of the failure. This value depends on the system the application is running on and can range between a few seconds to even one hour. However, for the systems we analyzed, in general, the time delay is very low and for our model we consider that is negligible compared to the checkpointing time. We add the waste of C seconds for each predicted failure, which happens every $\frac{MTTF}{N}$ seconds. After adding this waste equation (5) becomes:

$$W_{min}^{recall} = \sqrt{\frac{2C(1-N)}{MTTF}} + \frac{(R+D)}{MTTF} + \frac{CN}{MTTF}. \tag{6}$$

In the ideal case, when N=1, the minimum waste is equal to the time to checkpoint right before every failure and the time to restart after every failure. The formula assumes a perfect precision. In case the precision is P, the waste value must also take into consideration the cases when the prediction is wrong. The predicted faults happen every $\frac{MTTF}{N}$ seconds and they represent P of total predictions. This means that the rest of (1-P) false positives predictions will happen every $\frac{P}{1-P}\frac{MTTF}{N}$ seconds. Each time a false positive is predicted, a checkpointing is taken that must be added to the total waste from equation (6).

$$W_{min}^{recall} = \sqrt{\frac{2C(1-N)}{MTTF}} + \frac{(R+D)}{MTTF} + \frac{CN}{MTTF} + \frac{CN(1-P)}{P \cdot MTTF} \tag{7}$$

As an example, we consider the values used by [34] to characterize current systems: R = 5, D = 1 in minutes, and we study two values for the time to checkpoint: C=1 minute and from [25] C=10 seconds. We computed the gain from using the prediction offered by our hybrid method with different precision and recall values and for different MTTFs. Table IV presents the results. The first four cases present numbers from real systems and checkpointing strategies. Interestingly, for future systems with a MFFT of 5 hour, if the prediction can provide a recall over 50%, then the wasted time decreases by more than 20%. We plan to combine a checkpointing strategy with our prediction and study its effectiveness in real HPC systems.

C	Precision	Recall	MTTF for the whole system	Waste gain
1min	92	20	one day	9.13%
1min	92	36	one day	17.33%
10s	92	36	one day	12.09%
10s	92	45	one day	15.63%
1min	92	50	5h	21.74%
10s	92	65	5h	24.78%

TABLE IV
PERCENTAGE WASTE IMPROVEMENT IN CHECKPOINTING STRATEGIES

VII. CONCLUSION

This paper investigates a novel way of analyzing log files from large-scale systems, by combining two different analysis techniques: data-mining and signal processing; and using the advantages given by both. We use signal analysis concepts for shaping the normal behavior of each event type and of the whole system and characterizing the way faults affect them. The models we use are more realistic in that they take into account the different behavior of the events during failure-free execution and when failures occur. At the same time we use data mining algorithms for analyzing the correlations between these behaviors since these algorithms prove more suited in characterizing the interactions between different high dimensionality sets than the cross-correlation function offered by signal analysis.

In our experiments we show that a more realistic model, like the one obtained with the hybrid method, influences the prediction results and in the end the efficacy of fault tolerance algorithms is improved. We investigated both the lag time between the prediction moment and the time of occurrence for the actual failure, taking into consideration the analysis time, and we concluded that the proposed model could allow proactive actions to be taken. Moreover, since the location of an error in an important part of a prediction system, we included in our prediction location analysis and studied its impact on the results. We will focus in the future on a more detailed analysis of different error types for which our system has a low recall. Also, we plan to study to a wider extent, the practical way the prediction system influences current fault tolerance mechanisms.

ACKNOWLEDGMENTS

This work was supported in part by the DOE 9J-30281-0008A grant; and by the INRIA-Illinois Joint Laboratory for Petascale Computing.

REFERENCES

[1] N. Taerat, Y. Zhang, A. Sivasubramaniam, M. Jette, R. Sahoo: BlueGene/L Log Analysis and Time to Interrupt Estimation. International Conference on Availability, Reliability and Security, pp.173-180, 2009

[2] L. Di Jorio, A. Laurent, M. Teisseire: Mining frequent gradual itemsets from large databases. International Conference on Intelligent Data Analysis, IDA09, 2009

[3] A. Laurent , B. Negrevergne, N. Sicard, and A. Termier: PGP-mc: Towards a Multicore Parallel Approach for Mining Gradual Patterns Database Systems for Advanced Applications, volume 5981, pp 78-84, 2010

[4] A. Gainaru, F. Cappello, W. Kramer: Taming of the Shrew: Modeling the Normal and Faulty Behavior of Large-scale HPC Systems International Parallel and Distributed Processing Symposium, 2012

[5] Y. Liang: BlueGene/L Failure Analysis and Prediction Models. Proceedings of the International Conference on Dependable Systems and Networks, pp 425 - 434, 2006

[6] R. K. Sahoo et al: Critical Event Prediction for Proactive Management In Large-scale Computer Clusters. International conference on Knowledge discovery and data mining, pp 426-435, 2003

[7] N. Yigitbasi et al: Analysis and Modeling of Time-Correlated Failures in Large-Scale Distributed Systems. IEEE/ACM International Conference on Grid Computing, pp 65-72, 2010

[8] J. G. Lou et al: Mining Dependency in Distributed Systems through Unstructured Logs Analysis ACM SIGOPS Volume 44 Issue 1, January 2010

[9] W. Xu et al: Online System Problem Detection by Mining Patterns of Console Logs IEEE International Conference on Data Mining, pp 588-597, 2009

[10] Z. Zheng et al: A Practical Failure Prediction with Location and Lead Time for Blue Gene/P International Conference on Dependable Systems and Networks Workshops, pp 15-22, 2010

[11] J. Gu et al: Dynamic Meta-Learning for Failure Prediction in Large-Scale Systems: A case Study International Conference on Parallel Processing, pp 157-164, 2008

[12] A. Gainaru, F. Cappello, S. Trausan-Matu, W. Kramer: Adaptive Event Prediction Strategy with Dynamic Time Window for Large-Scale HPC Systems. System Log Analysis with Machine Learning Workshop, SLAML, 2011

[13] R. Ren et al: LogMaster: Mining Event Correlations in Logs of Large-scale Cluster Systems CoRR abs/1003.0951, 2010

[14] N. Nakka et al: Predicting Node Failure in High Performance Computing Systems from Failure and Usage Logs IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems, 2011

[15] A. Gainaru, F. Cappello, S. Trausan-Matu, W. Kramer: Event log mining tool for large scale HPC systems. International Conference on Parallel Processing Euro-Par, volume 1, pp 52-64, 2011.

[16] R. Iyer et al: Automatic recognition of intermittent failures: An experimental study of field data. IEEE Transactions on Computers, 39:525537, 1990.

[17] A. Pecchia, D. Cotroneo, Z. Kalbarczyk, R. Iyer: Improving Log-Based Field Failure Data Analysis of Multi-Node Computing Systems International Conference on Dependable Systems and Networks (DSN), pp 97-108, 2011

[18] Z. Zheng, Z. Lan, B. Park, and A. Geist: System log preprocessing to improve failure prediction. International Conference on Dependable Systems and Networks, pp 572-577, 2009.

[19] A. Oliner et al: What Supercomputers Say: A Study of Five System Logs. International Conference on Dependable Systems and Networks, 2007

[20] E. Heien, D. Kondo, A. Gainaru, D. LaPine, W. Kramer, F. Cappello: I Modeling and Tolerating Heterogeneous Failures in Large Parallel Systems. International Conference for High Performance Computing, Networking, Storage and Analysis, 2011

[21] B. Schroeder et al: A large-scale study of failures in high-performance computing systems. International Conference on Dependable Systems and Networks, pages 249-258, June 2006

[22] R. C. Milton: An Extended Table of Critical Values for the Mann-Whitney (Wilcoxon) Two-Sample Statistic. Journal of the American Statistical Association, Volume 59, Issue 3, 1978

[23] National Center for Supercomputing Applications at the University of Illinois. www.ncsa.illinois.edu. Accessed on 2010.

[24] The computer failure data repository. http://cfdr.usenix.org. Accessed on 2010.

[25] L. B. Gomez et al: FTI: high performance Fault Tolerance Interface for hybrid systems. International Conference for High Performance Computing, Networking, Storage and Analysis, 2011.

[26] G. Zheng et al: FTC-Charm++: An In-Memory Checkpoint-Based Fault Tolerant Runtime for Charm++ and MPI. International Conference on Cluster Computing CLUSTER, pp 93-103, 2004

[27] R. Agarwal et al: Rebound: Scalable Checkpointing for Coherent Shared Memory. ACM SIGARCH Computer Architecture News, Volume 39 Issue 3, 2011

[28] A. Moody, G. Bronevetsky, K. Mohror, B. R. de Supinski: Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp.1-11, 2010

[29] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, P. Beckman: A Practical Failure Prediction with Location and Lead Time for Blue Gene/P Proceedings of the 2010 International Conference on Dependable Systems and Networks Workshops, pp 15-22, 2010

[30] C. Wang, F. Mueller, C. Engelmann, and S. Scott: Proactive process-level live migration in HPC environments. International Conference for High Performance Computing, Networking, Storage and Analysis, 2008.

[31] J. Gu, Z. Zheng, Z. Lan, J. White, and B. Park: Dynamic meta-learning for failure prediction in large-scale systems: A case study. International Conference on Parallel Processing, 2008.

[32] R. Sahoo and A. Oliner: Critical event prediction for proactive management in large-scale computer clusters. SIGKDD International Conference on Knowledge Discovery and Data mining , pp 426-435, 2003.

[33] M. Steinder and A. Sethi: A survey of fault localization techniques in computer networks. Science of Computer Programming, volume 53, issue 2, 2004

[34] F. Cappello, H. Casanova, and Y. Robert: Checkpointing vs. migration for post-petascale supercomputers. International Conference on Parallel Processing, 2010