Ensemble of Bayesian Predictors and Decision Trees for Proactive Failure Management in Cloud Computing Systems

Qiang Guan, Ziming Zhang and Song Fu
Department of Computer Science and Engineering
University of North Texas
Denton, Texas 76203, USA
Email: {QiangGuan, ZimingZhang}@my.unt.edu; Song.Fu@unt.edu

***Abstract*— In modern cloud computing systems, hundreds and even thousands of cloud servers are interconnected by multi-layer networks. In such large-scale and complex systems, failures are common. Proactive failure management is a crucial technology to characterize system behaviors and forecast failure dynamics in the cloud. To make failure predictions, we need to monitor the system execution and collect health-related runtime performance data. However, in newly deployed or managed cloud systems, these data are usually unlabeled. Supervised learning based approaches are not suitable in this case. In this paper, we present an unsupervised failure detection method using an ensemble of Bayesian models. It characterizes normal execution states of the system and detects anomalous behaviors. After the anomalies are verified by system administrators, labeled data are available. Then, we apply supervised learning based on decision tree classifiers to predict future failure occurrences in the cloud. Experimental results in an institute-wide cloud computing system show that our methods can achieve high true positive rate and low false positive rate for proactive failure management.**

***Index Terms*— Cloud computing, Dependability assurance, Failure prediction, Unsupervised and supervised learning, Ensemble of Bayesian models, Decision tree.**

I. Introduction

As data centers and cloud computing systems are growing more and more complex, they are also changing dynamically due to the addition and removal of system components, changing execution environments, frequent updates and upgrades, online repairs, mobility of devices, and the system and network complexity. Classical reliability theory and conventional methods do rarely consider the actual state of a system and are therefore not capable to reflect the dynamics of runtime systems and failure processes. Such methods are typically useful in design for long term or average behavior modeling and comparative analysis.

With ever-growing complexity and dynamicity of cloud computing systems, proactive failure management is an effective approach to enhance system dependability [1]. Failure prediction is the key to such techniques. It forecasts future failure occurrences in the cloud using runtime execution states of the system and the history information of observed failures. It provides valuable information for resource allocation, computation reconfiguration and system maintenance [2]. In contrast to classical reliability methods, failure prediction is based on runtime monitoring and a variety of models and methods that use the current state of a system and the past experience as well.

Most of the existing failure prediction methods are based on statistical learning techniques [3]. They use supervised learning models to approximate the dependency of failure occurrences on various performance features [1], [4]–[6]. The underlying assumption of those methods is that the training dataset is labeled, i.e. for each data point used to train a predictor, the designer knows if it is corresponded to a normal execution state or a failure. However, the labeled data are not always available in real-world cloud computing systems, especially for newly deployed or managed systems. How to accurately forecast failure occurrences in such systems is challenging.

In this paper, we propose two learning approaches that use Bayesian methods and decision trees to predict failure dynamics in cloud computing systems. First, we tackle the problem from an anomaly detection viewpoint, for which we introduce an ensemble of Bayesian models. It works in an unsupervised learning manner and deals with unlabeled datasets. This model estimates the probability distribution of runtime performance data collected by health monitoring tools when cloud servers perform normally. After the detected anomalous behaviors are verified by the system administrators, labels are added to these data points. The second method is supervised decision tree classifiers, which explore the labeled data and predict failure occurrences in the cloud. We implement a prototype of our proactive failure management framework and evaluate its performance on an institute-wide cloud computing system. Experimental results show that our proposed methods can forecast failure dynamics with high accuracy.

The rest of this paper is organized as follows: Section II describes an ensemble of Bayesian models for unsupervised failure detection. Section III presents decision tree classifiers for supervised failure prediction. Experimental

Manuscript received August 15, 2011; accepted October 5, 2011.

doi:10.4304/jcm.7.1.52-61

results are included in Section IV. Section V discusses the related works. Conclusion and remarks on future work are presented in Section VI.

II. Unsupervised Failure Detection by Ensemble of Bayesian Models

A recent system reliability study on a 512-node LLNL ASC White machine showed that the mean time to failure of a node was about 160 days [7]. We may label the runtime health related data with one of two classes, Class 0 for normal behavior and Class 1 for situations with failures. Then, Class 1 is very rare compared with Class 0. For Class 1, there many not be enough data available to allow a supervised learning algorithm to estimate a good probability model for that class. In addition, data from the rare class may be incomplete because of some collection problems. This is especially true when a node suddenly crashes which leaves no time for the monitoring tools to retrieve and save its performance data.

An alternative to supervised learning that tackles the unbalanced dataset is to build a probabilistic model of the majority class and use failure detection methods to cluster and characterize health-related data. Failure detection algorithms classify data as normal or not based on a probability model of normal behavior. A failure is a data point to which the majority class model assigns a very low probability of occurring. A failure detection algorithm can build a probability model and learn its parameters in (1) an unsupervised manner in which both normal and failure data are learned (This approach assumes the failure data are too rare to affect the model parameters significantly.) or (2) a semisupervised manner in which only data of normal behavior are learned to construct a model. Semisupervised learning is in general preferable because it may generate a more accurate model for the normal class. However, in real-world cases most collected data are not labeled and the failure class is a rare one. As a result, the unsupervised learning approach is more practical and useful.

A. Dimensionality Reduction

Health-related data are collected across a cloud system and the data transformation component assembles the data into a uniform format. A feature in the dataset refers to any individual measurable variable of a cloud server being monitored. It can be the system or user-level CPU utilization, CPU idle time, memory utilization, volume of I/O operations, and more. There may be hundreds or even thousands of features for large data centers and cloud systems. The large number of performance metrics that are measured and the overwhelming volume of data collected by health monitoring tools make the data model extremely complex. Moreover, the existence of interacting metrics and external environmental factors introduce measurement noises in the collected data.

To make failure detection tractable and practical and yield high detection accuracy, we apply dimensionality reduction, which transforms the collected health data to a new feature space with only the most relevant attributes preserved [8]. The resulting data are input to an ensemble of Bayesian submodels which generates probability models of the data and performs failure detection.

1) Relevance Deduction: For probabilistic modeling and failure detection, large feature sets introduce high dimensionality. Moreover, with unsupervised learning, high dimensionality makes features less distinctive to each other. Usually, features with high similarity impede the performance of grouping and characterizing in unsupervised learning. Similarity of features can be evaluated by using the mutual information. In the following discussions, we denote discrete random variables of different features by $X_1, X_2, \cdots, X_n$.

The mutual information for two features is defined as follow.

$$I(X_i; X_j) = \sum_{x_i} \sum_{x_j} p(x_i, x_j) log \frac{p(x_i, x_j)}{p(x_i)p(x_j)} \qquad \text{(II.1)}$$

It quantifies how much information is shared between X_i and X_j. Mutual information has been widely used as a feature selection method [9]. $I(X_i, X_j)$ can measure the goodness of a term globally between two features. Those feature pairs with high co-relevance have large mutual information. If X_i and X_j are independent, their mutual information has a minimum value as zero. If X_i and X_j are the same feature, their mutual information has a maximum value as one. The objective of utilizing mutual information is to reduce the relevance of a selected subset of features. Therefore, a feature, which has high mutual information with other features, should be excluded from the subset. The index for evaluation of the relevance is defined as follow.

$$Index(X_i) = \sum_{j=1}^{i-1} I(X_i; X_j) + \sum_{j=i+1}^{n} I(X_i; X_j) \qquad \text{(II.2)}$$

It can be proved that two features with linear relation have high index, which shows evidently high relevance.

2) Redundancy Deduction: For feature selection, it has been known that the combination of individually independent features does not necessarily lead to a good performance on grouping and probabilistic analysis. The existence of redundant dimensions can bring problems. PCA (principle component analysis) has been widely used and proved to be powerful in eliminating redundancy of a subset of features [10]. By using PCA, a matrix A is generated from the health-related dataset. It is constructed as an $m \times n$ matrix, where m is the number of instances in the dataset and n refers to the number of attribute values. Then, the covariance matrix of A is calculated, which is denoted by C. Each element in C is defined as

$$c_{i,j} = covaraince(a_{*,i}, a_{*,j}), \qquad \text{(II.3)}$$

where $c_{i,j}$ is the covariance of the ith and jth attributes of the health data. The covariance of two attributes measures the extent to which the two attributes vary together.

PCA performs a coordinate rotation that aligns the transformed axes with the directions of maximum variance. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. This means if the summation of the principal components possesses most of variability of the whole dataset, such as 90% or above, the components, which contribute to the most of the variability, can take the place of the original dataset. These components constitute the selected subset.

B. Ensemble of Bayesian Submodels

In order to detect possible failures, we analyze the health-related data and construct statistical models. A probabilistic model f is chosen for the data with reduced dimensionality. It takes a data point d as input and outputs a probability for that data point. The parameters of f are learned from training data in an unsupervised manner. Then d is detected as an failure if and only if $f(d) < t$, where t is a threshold, whose value can be determined based on assumptions of the rarity of failure data or from learning experiments if failures are labeled in the collected health-related data. A data point is labeled as normal or failure based on its probability of appearance as a normal data point.

To construct the probabilistic model and assure high detection accuracy, we develop an ensemble of Bayesian submodels to represent a multimodal probability distribution. Each submodel is a nonparametric data model, in which no single simple parameterized probability density is assumed. Its probability distribution is determined from the frequency counts of the training data. Each submodel has an estimated prior probability $p(m)$, where m is the submodel index. The probability estimate assigned to a data point d is

$$p(d) = \sum_{m \in submodels} p(d|m)p(m), \qquad \text{(II.4)}$$

where $p(d|m)$ is the probability that submodel m generates data point d. Therefore, all submodels contribute to the probability of each data point. Also, a data point d is assigned to a submodel m with probability $p(m|d)$, whose value can be determined by using the Bayes' rule. This approach allows our model to fit the collected data better when it is unknown which submodel should be used to characterize the probability of a data point. After relevance reduction and redundancy reduction as described in Section II-A, the features of a data point are conditionally independent in each submodel. Therefore, for a data point d with k features after dimensionality reduction, the conditional probability of the data point on a submodel m follows

$$p(d|m) = \prod_{i=1}^{k} p(d_i|m). \qquad \text{(II.5)}$$

We use discrete Bayesian submodels, where the values of each feature are placed in a finite number of intervals. The discrete indexes of intervals replace the original value of a feature. We adopt the Bayesian expectation-maximization (EM) algorithm to estimate the probability that a feature d_i takes a given value v based on the counted frequency in the training dataset.

$$p(d_i = v|m) = count(d_i = v \text{ and } m)/count(m), \qquad \text{(II.6)}$$

$$p(m) = count(m) \; / \sum_{n \in submodels} count(n), \qquad \text{(II.7)}$$

where $count(\cdot)$ is the number of data points in the training dataset that satisfy a specified condition. In Equations (II.6) and (II.7), $count()$ is calculated as

$$count(d_i = v \text{ and } m) = \sum_{d \in trainingset} p(m|d) \cdot I(d_i, v), \qquad \text{(II.8)}$$

$$count(m) = \sum_{d \in trainingset} p(m|d), \qquad \text{(II.9)}$$

$$I(d_i, v) = \begin{cases} 1, & \text{if } d_i = v \\ 0, & \text{if } d_i \neq v \end{cases} \qquad \text{(II.10)}$$

To train the ensemble model, we choose the number of submodels. We initialize the model by assigning data points randomly to submodels. The Expectation-Maximization algorithm is performed to determine the submodel and conditional data probabilities, $p(m)$ and $p(d|m)$ respectively. Then, Equation (II.4) is applied to calculate the data probability. The EM algorithm proceeds in rounds of an expectation step (E-step) followed by a maximization step (M-step). For a data point d, the E-step calculates the probability of each submodel m generating d. In the calculation, we use the Bayes' rule, $p(m|d) = p(d|m)p(m)/p(d)$, where the right-hand side is computed by Equations (II.4) and (II.5). After an E-step completes, an M-step updates probabilities $p(d_i|m)$ and $p(m)$ by Equations (II.6) and (II.7). It maximizes the likelihood of the model given the expected probability. The E-step and M-step continue until the likelihood does not change.

III. Supervised Failure Prediction by Decision Trees

The failure detection method based on unsupervised learning presented in the preceding section identifies anomalous behaviors in a cloud system. The anomalies are reported to the system administrations for verification. If they are confirmed as failures or as normal, the corresponding data points are labeled. As the cloud system continues operation, more data points will be labeled. These labeled data provide valuable information about the system states under failures. They should be exploited in failure prediction. In this section, we present a supervised learning based method for failure prediction. An ensemble of decision tree classifiers is proposed to forecast failure occurrences using health-related performance data with labels.

Metric	Description
CPU Statistics	
PROC/S	Total number of tasks created per second
Cswch/s	Total number of context switches per second.
%user	Percentage of CPU utilization that occurred while executing at the user level (application). Note that this field includes time spent running virtual processors.
%nice	Percentage of CPU utilization that occurred while executing at the user level with nice priority.
%system	Percentage of CPU utilization that occurred while executing at the system level (kernel). Note that this field includes time spent servicing interrupts and soft IRQs.
%iowait	Percentage of time that the CPU or CPUs were idle during which the system had an outstanding disk I/O request.
%idle	Percentage of time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request.
Intr/s	Shows the total number of interrupts received per second by the CPU.
SWAP Statistics	
PSWPIN/s	Total number of swap pages the system brought in per second.
PSWPOUT/s	Total number of swap pages the system brought out per second.
PGPGIN/s	Total number of kilobytes the system paged in from disk per second.
PGPOUT/s	Total number of kilobytes the system paged out to disk per second.
Fault/s	Number of page faults (major+minor) made by the system per second.
Majfit/s	Number of major faults the system made per second, those which have required loading a memory page from disk.
I/O Requests	
Tps	Total number of transfers per second that were issued to physical devices. A transfer is an I/O request to a physical device. Multiple logical requests can be combined into a single I/O request to the device. A transfer is of indeterminate size.
Rtps	Total number of read quests issued to physical devices.
Wtps	Total number of write requests per second issued to physical devices.
Bread/s	Total amount of data read from the devices in blocks per second. Blocks are equivalent to sectors with 2.4 kernels and newer and therefore have a size of 512 bytes. Older kernels, a block is of indeterminate size.
Bwrtn/s	Total amount of data written to devices in blocks per second.
Rd_sec/s	Number of sectors read from the device. The size of a sector is 512 bytes.
Wr_sec/s	Number of sectors written to the device. The size of a sector is 512 bytes.
Memory Page Statistics	
Frmpg/s	Number of memory pages freed by the system per second. A negative value represents a number of pages allocated by the system. Note that a page has a size of 4 kB or 8kB according to the machine architecture.
Bufpg/s	Number of additional memory pages used as buffers by the system per second. A negative value means fewer pages used as buffers by the system.
Campg/s	Number of additional memory pages cached by the system per second. A negative value means fewer pages used as buffers by the system.
KBMEMFREE	Amount of free memory available in kilobytes.
KBMEMUSED	Amount of used memory in kilobytes. This does not take into account memory used by the kernel itself.
%memused	Percentage of used memory.
KBBUFFERS	Amount of memory used as buffers by the kernel in kilobytes.
KBCACHED	Amount of memory used to cache data by the kernel in kilobytes.
KBSWPFREE	Amount of free swap space in kilobytes.
KBSWPUSED	Amount of used swap space in kilobytes.
%SWPUSED	Percentage of used swap space.
KBSWPCAD	Amount of cached swap memory in kilobytes. This is memory that once was swapped out, is swapped back in but still also is in the swap area (if memory is needed it doesn't need to be swapped out again because it is already in the swap area. This saves I/O).
Network Statistics	
RXPCK/s	Total number of packets received per second.
RXBYT/S	Total number of bytes received per second.
TXPCK/S	Total number of packets transmitted per second.
TXBYT/S	Total number of bytes transmitted per second.
RXCMP/s	Number of compressed packets received per second
RXERR/s	Total number of bad packets received per second.
TXERR/s	Total number of errors that happened per second while transmitting packets.
COLL/S	Number of collisions that happened per second while transmitting packets.
RXDROP/s	Number of received packets dropped per second because of a lack of space in Linux buffers.
TXCARR/s	Number of carrier-errors that happened per second while transmitting packets.
RXFRAM/s	Number of frame alignment errors that happened per second on received packets.
RXFIFO/s	Number of FIFO overrun errors that happened per second on received packets.
TXFIFO/s	Number of FIFO overrun errors that happened per second on transmitted packets.
TOTSCK	Total number of sockets used by the system.
TCPSCK	Number of TCP sockets currently in use.
UDPSCK	Number of UDP sockets currently in use.
RAWSCK	Number of RAW sockets currently in use.
IP-FRAG	Number of IP fragments currently in use.
RXMCST/s	Number of multicast packets received per second.

Figure 1. Runtime performance data collected by health monitoring tools in an institute-wide cloud computing system.

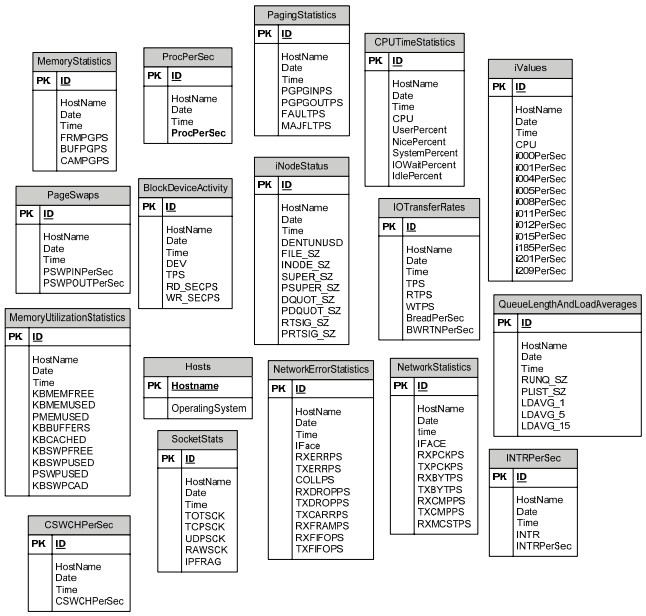

Figure 2. Database schema for storing performance data.

A. Learning Decision Trees

A decision tree is a hierarchical nonparametric model with local regions identified in a sequence of recursive splits [11]. A decision tree is composed of internal decision nodes and terminal leaves. Each decision node n implements a test function $f_n(d)$ with discrete outcomes labeling the branches. When a test hits a leaf node, the classification labeled on the leaf is output. There are many possible machine learning approaches for failure prediction. While decision trees are not always the most competitive classifiers in terms of prediction, they enjoy the crucial advantages of a fast localization of the region covering an input and yielding human interpretable results, which is important if the method is to be adopted by real cloud operators.

Learning a decision tree involves deciding which split to make at each node, and how deep the tree should be. Let X denote the feature set. For binary classification, the class label $\in \{0, 1\}$, where 1 denotes a failure and 0 normal. The root node of the decision tree contains all of the health-related performance data. At each node, the dataset is split according to the values of one particular feature. Splits are selected in order to maximize the *gain* in information. This process continues until no further split is possible or the node contains only one class. After the tree is built, sub-branches with low overall gain value are pruned to avoid overfitting.

The goodness of a split is measured by *impurity*. A split is pure if after the split, for all branches, all the data taking a branch belong to the same class. We use *entropy* to quantify impurity. For a node n in the decision tree, the entropy is calculated by

$$H(n) = -\sum_{i=1}^{j} p_n^i log_2 p_n^i, \qquad \text{(III.1)}$$

where j is the number of classes and $j = 2$ representing the *normal* and *failure* classes, and p_n^i is the probability of class C_i (0 or 1), given that a data point reaches node n. Then the gain function G for feature x_i at node n is defined as

$$G(x_i, n) = H(n) - H(x_i, n), \qquad \text{(III.2)}$$

where $H(x_i, n)$ denotes the sum of entropy of children nodes after making the split based on feature x_i.

The algorithm for building a decision tree works as follows. It begins with the root node which includes all the features. For each feature x_i, the gain value from splitting on x_i is calculated using Equation (III.2). Then, the feature x_{best} with the highest gain value is selected. A decision node that splits on x_{best} is created. Repeat the preceding process on the sublists obtained by splitting on x_{best} and add those nodes as children nodes.

B. Failure Prediction

We grow the decision tree full until all leaf nodes are pure and we have zero training error. The tree might be too deep and complex. We then find subtrees that cause overfitting and we prune them. From the initial labeled health-related data, we set aside a *pruning dataset*, unused in training. For each subtree, we replace it by a leaf node labeled with the training data points covered by the subtree. If the leaf node does not perform worse than the subtree on the pruning set, we prune the subtree and keep the leaf node because the additional complexity of the subtrees is not necessary; otherwise, we keep the original subtree.

After the decision tree is built and pruned, we exploit it for failure prediction. For a new and unlabeled data point, the failure predictor traverses the decision tree from the root. At each internal decision node, the predictor reads the value of the feature associated with the node from the input data point and selects a path to a child node accordingly. This process is repeated until a leaf node is hit. The predictor outputs the label (normal or failure) of the leaf. To achieve high prediction accuracy, we apply boosting to the decision tree classifiers and obtain an ensemble of trees. A voting is performed and the majority is selected as a failure prediction decision.

IV. Performance Evaluation

We have designed a proactive failure management framework, in which runtime performance data of a cloud computing system are collected by health monitoring tools and failure events are forecast by an ensemble of Bayesian predictors and decision tree classifiers. As a proof of concept, we implement a prototype of the failure manager. In this section, we evaluate the performance of our framework for autonomic failure management in a production cloud computing system.

TABLE I.
NORMALIZED MUTUAL INFORMATION MATRIX FOR CPU TIME STATISTICS FEATURES.

Features	proc/s	%user	%system	%iowait	%idle
proc/s	0	0.681	0.745	1.000	0.089
%user	0.681	0	0.656	0.714	0.042
%system	0.745	0.656	0	0.609	0.127
%iowait	1.000	0.714	0.609	0	0.118
%idle	0.089	0.042	0.127	0.118	0

TABLE II.
NORMALIZED MUTUAL INFORMATION MATRIX FOR MEMORY UTILIZATION STATISTICS FEATURES.(F1:KBMEMFREE; F2:KBMEMUSED; F3:%MEMUSED; F4:KBBUFFERS; F5:KBCACHED; F6:KBSWPFREE; F7:KBSWPUSED; F8:%SWPUSED; F9:KBSWPCAD; F10:PGPGOUT/S; F11:FAULT/S)

Features	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
F1	0	0.687	0.711	0.593	0.661	0.054	0.019	0.041	0.038	0.094	0.071
F2	0.687	0	0.862	0.602	0.771	0.040	0.037	0.036	0.040	0.088	0.067
F3	0.711	0.862	0	0.515	0.627	0.038	0.022	0.028	0.041	0.090	0.069
F4	0.593	0.602	0.515	0	0.609	0.078	0.051	0.078	0.055	0.047	0.034
F5	0.661	0.771	0.627	0.609	0	0.075	0.080	0.072	0.096	0.052	0.041
F6	0.054	0.040	0.038	0.078	0.075	0	0.728	1.000	0.598	0.017	0.079
F7	0.019	0.037	0.022	0.051	0.080	0.728	0	0.789	0.727	0.017	0.081
F8	0.041	0.036	0.028	0.078	0.072	1.000	0.789	0	0.612	0.016	0.073
F9	0.038	0.040	0.041	0.055	0.096	0.598	0.727	0.612	0	0.015	0.080
F10	0.094	0.088	0.090	0.047	0.052	0.017	0.017	0.016	0.015	0	0.732
F11	0.071	0.067	0.069	0.034	0.041	0.079	0.081	0.073	0.080	0.732	0

A. Experimental Environment

We test our failure manager in an institute-wide cloud computing environment which consists of 11 Linux server clusters on campus. Two clusters contain 116 servers each, while others host 32, 16, 22, 20, 10 and 8 servers in them. In total, there are 362 high-performance servers in the cloud. Among all the servers, 89.2% of them were up most of the time from September 5, 2009 to December 28, 2010. The cloud servers are equipped with 4 to 8 Intel Xeon or AMD Opteron cores and 2.5 to 16 GB of RAM. Within each cluster, servers are interconnected by gigabit Ethernet switches (The two clusters with 116 servers each are equipped with 2 gigabit Myrinets). Connections between clusters are through gigabit Ethernet. The two largest clusters are equipped with SANs.

Faculty and students submit application jobs using a web based interface. The cloud resource management software chooses server node(s) to instantiate virtual machines for the jobs. Typical applications running on the cloud include large database applications, molecular dynamics simulations, genome and proteome analysis, chemical kinetics simulations, materials and metallodrugs property analysis, and more. The cloud is also open to institute students to execute their sequential and parallel programs.

B. Health Data Collection and Dimensionality Reduction

We use sysstat [12] to collect runtime performance data on each server in the cloud. The values of 83 performance metrics are recorded every five minutes. They cover the statistics of every components of each cloud server, including CPU usage, process creation, task switching activity, memory and swap space utilization, paging and page faults, interrupts, network activity, I/O and data transfer, power management, and more. Figure 1 shows the major features. The collected data are pushed to master servers in the cloud for system-wide health analysis.

The collected data are *cleaned* first. Missing values of features in the data are filled by the average of two adjacent values. Then the data are parsed and transformed into a uniform format. A C# program is written to parse the collected data. The parsing program is written using regular expressions and is about 1,500 lines of code. After getting parsed, the data are formatted into the CSV format and then inserted into a database. Figure 2 depicts the database schema.

After the health-related data are cleaned, the training dataset is passed to the feature selection component. A mutual information-based feature selection algorithm is used to choose independent features that capture most information. Table I and Table II list the mutual information for each pair of features in the CPU time statistics category and memory utilization statistics category respectively. Figure 3 shows the normalized relevance of the sixteen CPU and memory related features. Based on the results, eight features, $\%user$, $\%system$, $\%idle$, $kbbbuffer$, $pgpgout/s$, $fault/s$, $rxpck/s$, $txpck/s$, and tps, whose dependency is below the threshold, are selected by the relevance deduction process.

Then, the PCA algorithm is applied to reduce redundancy among the selected eight features. The results are shown in Figure 4. From the figure, we can see the first and second principal components account for more than 94.54% of variability of the original dataset. Therefore, we reduce the dimension further to two.

C. Failure Prediction Performance

We measure the *true positive rate* and *false positive rate* in failure detection and prediction.

$$true\ positive\ rate = \frac{true\ positives}{true\ positives + false\ negatives},$$

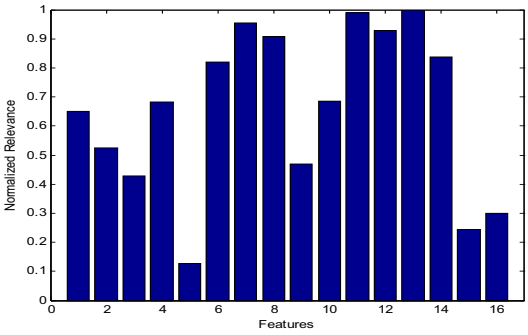

Figure 3. Mutual Information of CPU and Memory Related Features. The 16 features are F1:represent proc/s; F2: %user; F3: %system; F4: %iowait; F5: %idle; F6: kbmemfree, F7: kbmemused; F8: memused; F9: kbbuffers; F10: kbcached; F11: kbswpfree; F12: kbswpused; F13: %swpused; F14: kbswpcad; F15: pgpgout/s F16: fault/s.

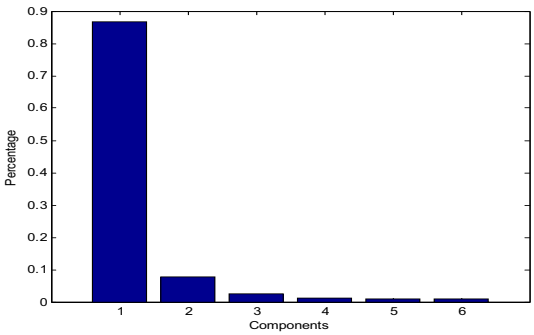

Figure 4. Redundancy Deduction by Principle Component Analysis.

$$false\ positive\ rate = \frac{false\ positives}{false\ positives + true\ negatives}.$$

We conduct a ten-fold cross validation to obtain the average prediction performance because of the randomized initialization performed by the failure detectors in Section II-B. In the training set, 10% of the normal data points are used for testing of cross-validation. We calculate the true positive rates and false positive rates and plot ROC curves, which show prediction accuracy with different threshold values. Figures 5 and 6 show the results for failure detection by an ensemble of Bayesian submodels and failure prediction by decision trees. Optimal performance would be at the top left of each figure, that is with high true positive rate and low false positive rate. Each point in Figure 5 is the average taken over all folds of ten-fold cross validation. From the figures, we can see both the ensemble of Bayesian submodels and the decision tree classifiers are able to predict failures in the health-related dataset while avoiding forecasting too many normal instances as failures. Moreover, by using all of the eight significant features selected by the relevance deduction procedure, the prediction accuracy is better than that by using only two most significant features chosen from both relevance and redundancy reduction procedures, while the computational overhead is not increased by much.

V. Related Works

Failure management is a crucial technique for understanding emergent, system-wide phenomena and self-managing resource burdens for system-level dependability and productivity assurance.

The conventional method for failure management and fault tolerance relies on checkpointing/restart mechanisms, which periodically save a snapshot of a system to a stable storage and use it to recover the system from failures reactively; see [13] for a comprehensive review and [14]–[17] for examples. However, this method does not prevent systems from failures, and work loss is inevitable due to its rollback process [13]. Moreover, checkpointing a job in a large-scale system could incur significant overhead. The LANL study [18] estimates the checkpointing overhead based on the current techniques to run a 100 hour job (without failure) by an additional 151 hours in petaflop systems. As a result, frequent periodic checkpointing often proves counter-effective.

As the scale and complexity of production systems continue to grow, research on system dependability has recently shifted onto failure prediction and proactive management technologies [5], [19]–[29]. Recent studies [30]–[36] apply execution migration techniques to enhance resource management by avoiding possible failures. They demonstrate the feasibility of exploiting proactive management methods for dependability assurance in networked computer systems. In this work, we exploit learning technologies to characterize system behaviors and forecast failure occurrences in cloud computing environments. Statistical learning has also been widely applied to resource management in computer systems and networks [37], [38].

To realize proactive failure management, it is imperative to understand the characteristics of failure behaviors. Research in [39]–[42] studied event traces collected from clusters and supercomputers. They found that failures are common in large-scale systems and their occurrences are quite dynamic, displaying uneven distributions in both time and space domains. There exist the time-of-day and day-of-week patterns in long time spans [39], [41]. Weibull distributions were used to model time-between-failure in [43]. Failure events, depending on their types, display strong spatial correlations: a small fraction of nodes may experience most of the failures in a coalition system [41] and multiple nodes may fail almost simultaneously [40].

Noticeable progress has been made on failure management research, and failure analysis [39]–[42] reveals failure characteristics in high performance computing systems. Zhang et al. evaluated the performance implications of failures in large scale clusters [44]. Sahoo et al. [1] inspected the eventset within a fixed time window before a target event for repeated patterns to predict the failure event of all types. Liang et al. [19] profiled the time-between-failure of different failure types and applied a heuristic approach to detect failures by using a monitoring window of preset size. Mickens and Noble [4] assumed the independency of failures among compute nodes and

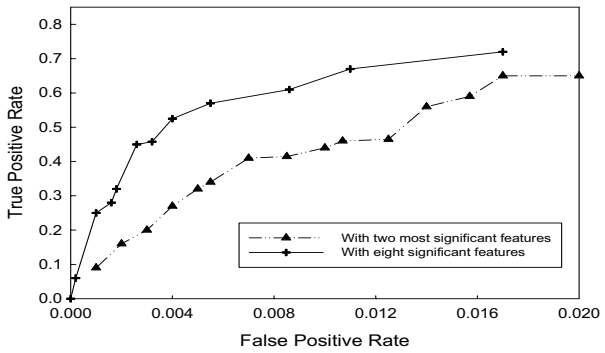

Figure 5. Performance on failure detection by the ensemble of Bayesian submodels.

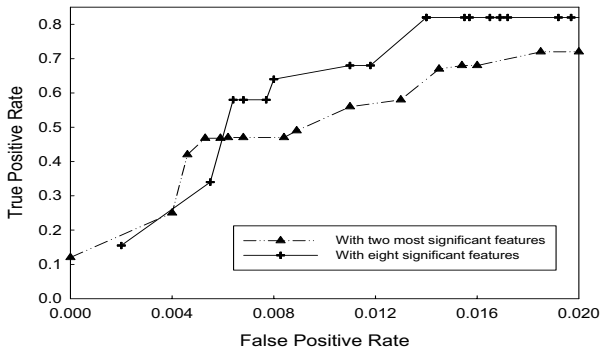

Figure 6. Performance on failure prediction by decision tree classifiers.

used the per-node uptime data to predict whether a failure might occur on that node in the next time window of fixed size. Fu and Xu [21] exploited the temporal and spatial correlations among failure events to predict the occurrence time of next failures in HPC systems. Gokhale and Trivedi [45] forecast the software reliability by using Markov chains to represent software architecture. Most of these works focused on improving the prediction accuracy, and few of them considered how to leverage their prediction results for resource management in practice.

Data mining and statistical learning technologies have received growing attention for failure diagnosis and diagnosis. They do not require a priori model or knowledge of failure distributions. Failure patterns are learned and discovered from normal system behaviors. They use the learned patterns to detect anomalous behaviors [46]. For example, the group at the Berkeley RAD laboratory applied statistical learning techniques for failure diagnosis in Internet services [47]. Similar techniques were applied to automate failure management in information technology systems [48]. Statistical approaches were studied in forecasting failure events on the BlueGene/L supercomputer [19].

VI. Conclusion

Large-scale and complex data centers and cloud computing systems are susceptible to software and hardware failures and administrators' mistakes, which significantly affect the system performance and management. In this paper, we present an integrated proactive failure management framework. At the initial stage of health monitoring and control, no labeled data are available. We propose to use an ensemble of Bayesian models to characterize normal states of the system and to detect anomalous behaviors in an unsupervised learning manner. After the anomalies are verified, either confirmed as failures or classified as normal, by the system administrators, labeled data are available. We explore these health-related data with labels and apply supervised learning based on decision tree classifiers to forecast future failure occurrences in the cloud. We implement a prototype of our proactive failure management system and test its performance in a production cloud computing environment. Experimental results show our methods can achieve high true positive rate and low false positive rate for failure prediction.

In this work, we use Bayesian classifiers and decision trees for failure detection and prediction. We plan to explore other advanced statistical learning techniques for proactive failure management. We also note that even with the most advanced learning algorithms the prediction accuracy could not reach 100%. As a remedy, reactive failure management techniques, such as checkpointing and redundant execution, can be exploited to deal with mis-predictions. As a future work, we will integrate these two failure management approaches and enhance the cloud dependability further.

Acknowledgment

We would like to thank the anonymous reviewers for their constructive comments and suggestions. This research was supported in part by U.S. NSF grant CNS-0915396 and LANL grant IAS-1103. A preliminary version of this paper was published in the proceedings of the 20th IEEE International Conference on Computer Communications and Networks (ICCCN), 2011 [49].

References

[1] R. K. Sahoo, A. J. Oliner, I. Rish, and et al., "Critical event prediction for proactive management in large-scale computer clusters," in *Proceedings of ACM International Conference on Knowledge Discovery and Data Dining (SIGKDD)*, 2003.

[2] A. J. Oliner, R. K. Sahoo, J. E. Moreira, and et al., "Fault-aware job scheduling for BlueGene/L systems," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[3] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys*, vol. 42, pp. 10:1–10:42, 2010.

[4] J. W. Mickens and B. D. Noble, "Exploiting availability prediction in distributed systems," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.

[5] S. Fu and C. Xu, "Quantifying event correlations for proactive failure management in networked computing systems," *Journal of Parallel and Distributed Computing*, vol. 70, no. 11, pp. 1100–1109, 2010.

[6] J. Gu, Z. Zheng, Z. Lan, J. White, E. Hocks, and B.-H. Park, "Dynamic meta-learning for failure prediction in large-scale systems: A case study," in *Proceedings of IEEE International Conference on Parallel Processing (ICPP)*, 2008.

[7] H. Song, C. Leangsuksun, and R. Nassar, "Availability modeling and analysis on high performance cluster computing systems," in *Proceedings of IEEE International Conference on Availability, Reliability and Security (ARES)*, 2006.

[8] J. Han, *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 2005.

[9] T. Cover and J. Thomas, *Elements of Information Theory*. John Wiley & Sons, 1991.

[10] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Wiley-Interscience, 2001.

[11] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[12] "sysstat," available at: http://sebastien.godard.pagesperso-orange.fr/.

[13] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375–408, 2002.

[14] O. Laadan and J. Nieh, "Transparent checkpoint-restart of multiple processes on commodity operating systems," in *Proceedings of USENIX Annual Technical Conference (USENIX)*, 2007.

[15] Y. Li and Z. Lan, "Fast restart mechanism for checkpoint/recovery protocols in networked environments," in *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2008.

[16] A. Agbaria and R. Friedman, "Model-based performance evaluation of distributed checkpointing protocols," *Performance Evaluation*, vol. 65, no. 5, pp. 345–365, 2008.

[17] G. Bronevetsky, D. J. Marques, K. K. Pingali, R. Rugina, and S. A. McKee, "Compiler-enhanced incremental checkpointing for openmp applications," in *Proceedings of ACM Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2008.

[18] I. Philp, "Software failures and the road to a petaflop machine," in *Proceedings of Symposium on High Performance Computer Architecture Workshop*, 2005.

[19] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. K. Sahoo, "BlueGene/L failure analysis and prediction models," in *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2006.

[20] F. Salfner, P. Tröger, and S. Tschirpke, "Cross-core event monitoring for processor failure prediction," in *Proceedings of IEEE International Conference on High Performance Computing & Simulation, Workshop on Dependable Multi-Core Computing(DMCC)*, 2009.

[21] S. Fu and C. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proceedings of ACM/IEEE Supercomputing Conference (SC)*, 2007.

[22] S. Fu and C. Xu, "Quantifying temporal and spatial correlation of failure events for proactive management," in *Proceedings of IEEE International Symposium on Reliable Distributed Systems (SRDS)*, 2007.

[23] S. S. Gokhale and K. S. Trivedi, "Analytical models for architecture-based software reliability prediction: A unification framework," *IEEE Transactions on Reliability*, vol. 55, no. 4, pp. 578–590, 2006.

[24] F. Salfner and M. Malek, "Using hidden semi-markov models for effective online failure prediction," in *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS)*, 2007.

[25] F. Salfner and M. Malek, "Proactive fault handling for system availability enhancement," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Dependable Parallel Distributed Network-centric Systems*, 2005.

[26] Z. Zhang and S. Fu, "Failure prediction for autonomic management of networked computer systems with availability assurance," in *Proceedings of IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems in conjunction with IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2010.

[27] Z. Zhang and S. Fu, "A hierarchical failure management framework for dependability assurance in compute clusters," *International Journal of Computational Science*, vol. 4, no. 4, pp. 313–326, 2010.

[28] Q. Guan and S. Fu, "auto-AID: A data mining framework for autonomic anomaly identification in networked computer systems," in *Proceedings of IEEE International Performance Computing and Communications Conference (IPCCC)*, 2010.

[29] Q. Guan, D. Smith, and S. Fu, "Anomaly detection in large-scale coalition clusters for dependability assurance," in *Proceedings of IEEE International Conference on High Performance Computing (HiPC)*, 2010.

[30] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for HPC with Xen virtualization," in *Proceedings of ACM International Conference on Supercomputing (ICS)*, 2007.

[31] A. Polze, P. Tröger, and F. Salfner, "Timely virtual machine migration for pro-active fault tolerance," in *Proceedings of IEEE International Workshop on Object/component/service-oriented Real-time Networked Ultra-dependable Systems (WORNUS), at IEEE 14th International Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC)*, 2011.

[32] S. Fu, "Failure-aware construction and reconfiguration of distributed virtual machines for high availability computing," in *Proceedings of IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, 2009.

[33] S. Chakravorty, C. Mendes, , and L. Kale, "Proactive fault tolerance in MPI applications via task migration," in *Proceedings of IEEE International Conference on High Performance Computing*, 2006.

[34] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in HPC environments," in *Proceedings of ACM/IEEE Conference on Supercomputing (SC)*, 2008.

[35] S. Fu, "Failure-aware resource management for high-availability computing clusters with distributed virtual machines," *Journal of Parallel and Distributed Computing*, vol. 70, no. 4, pp. 384–393, 2010.

[36] S. Fu and C. Xu, "Proactive resource management for failure resilient high performance computing clusters," in *Proceedings of IEEE International Conference on Availability, Reliability and Security (ARES)*, 2009.

[37] S. Muppala, X. Zhou, and L. Zhang, "Regression based multi-tier resource provisioning for session slowdown guarantees," in *IEEE International Performance Computing and Communications Conference (IPCCC)*, 2010.

[38] S. Muppala and X. Zhou, "Coordinated session-based admission control with statistical learning for multi-tier internet applications," *Journal of Network and Computer Applications, Elsevier*, vol. 34, no. 1, pp. 20–29, 2011.

[39] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance-computing systems," in *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2006.

[40] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, and M. Gupta., "Filtering failure logs for a BlueGene/L prototype," in *Proceedings of IEEE/IFIP*

International Conference on Dependable Systems and Networks (DSN), 2005.

[41] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2004.

[42] P. Yalagandula, S. Nath, H. Yu, P. B. Gibbons, and S. Sesha, "Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems," in *Proceedings of USENIX WORLDS*, 2004.

[43] T. Heath, R. P. Martin, and T. D. Nguyen, "Improving cluster availability using workstation validation," in *Proceedings of ACM Conference on Measurement and modeling of computer systems (SIGMETRICS)*, 2002.

[44] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo, "Performance implications of failures in large-scale cluster scheduling," in *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, 2004.

[45] S. S. Gokhale and K. S. Trivedi, "Analytical models for architecture-based software reliability prediction: A unification framework," *IEEE Trans. on Reliability*, vol. 55, no. 4, pp. 578–590, 2006.

[46] W. Peng, T. Li, and S. Ma, "Mining logs files for computing system management," in *Proceedings of IEEE International Conference on Automatic Computing (ICAC)*, 2005.

[47] A. Zheng, J. Lloyd, and E. Brewer, "Failure diagnosis using decision trees," in *Proceedings of IEEE International Conference on Automatic Computing (ICAC)*, 2004.

[48] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating instrumentation data to system states: a building block for automated diagnosis and control," in *Proceedings of USENIX Symposium on Opearting Systems Design and Implementation (OSDI)*, 2004.

[49] Q. Guan, Z. Zhang, and S. Fu, "Ensemble of bayesian predictors for autonomic failure management in cloud computing," in *Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN)*, 2011.

Qiang Guan is currently a Ph.D. candidate in Computer Science and Engineering at the University of North Texas. He received the BS degree in Communication Engineering from Northeastern University, China, in 2005, and the MS degree in Information Engineering from Myongji University, South Korea, in 2008. He was a Ph.D. student in Computer Science at Mexico Institute of Mining and Technology from January 2010 to July 2010. His research interests include failure modeling and management, dependable assurance, resource management, and virtual machines in distributed and cloud computing systems.

Ziming Zhang is currently a Ph.D. candidate in Computer Science and Engineering at the University of North Texas. He received the BS degree in Computer Science from Beihang University, China, in 2009. He was a Ph.D. student in Computer Science at Mexico Institute of Mining and Technology from August 2009 to July 2010. His research interests include energy efficiency, power profiling, power-aware resource management, dependable computing, and virtual machines in distributed and cloud computing systems.

Song Fu is currently an Assistant Professor in the Department of Computer Science and Engineering and the Director of the Dependable Computing Systems Laboratory at the University of North Texas. He was an Assistant Professor in Computer Science and Engineering at New Mexico Institute of Mining and Technology from August 2008 to July 2010. He received his Ph.D. degree in Computer Engineering from Wayne State University in 2008, M.S. degree in Computer Science from Nanjing University, China, in 2002, and B.S. degree in Computer Science from Nanjing University of Aeronautics and Astronautics, China, in 1999. His research interests include distributed, parallel and cloud systems, particularly in dependable computing, self-managing and reconfigurable systems, power management, energy-efficient systems, system reliability and security, resource management, and virtualization. His research projects have been sponsored by the U.S. National Science Foundation, Los Alamos National Laboratory, and the University of North Texas. He is a member of the IEEE and a member of the ACM.