Anomaly Localization in Large-Scale Clusters

Ziming Zheng, Yawei Li and Zhiling Lan

Department of Computer Science, Illinois Institute of Technology
Chicago, IL 60616, USA
{zzheng11, liyawei, lan}@iit.edu

***Abstract*— A critical problem facing by managing large-scale clusters is to identify the location of problems in a system in case of unusual events. As the scale of high performance computing (HPC) grows, systems are getting bigger. When a system fails to function properly, health-related data are collected for troubleshooting. However, due to the massive quantities of information obtained from a large number of components, the root causes of anomalies are often buried like needles in a haystack. In this paper, we present a localization method to automatically find out the potential root causes (i.e. a subset of nodes) of the problem from the overwhelming amount of data collected system-wide. System managers can focus on examining these potential locations, thereby significantly reducing human efforts required for anomaly localization. Our method consists of three interrelated steps: (1) *feature collection* to assemble a feature space for the system; (2) *feature extraction* to obtain the most significant features for efficient data analysis by applying the principal component analysis (PCA) algorithm; and (3) *outlier detection* to quickly identify the nodes that are "far away" from the majority by using the cell-based detection algorithm. Preliminary studies are presented to demonstrate the potential of our method for localizing anomalies in a computing environment where the nodes perform comparable tasks.**

I. Introduction

As the scale of high performance computing (HPC) grows, the new generation of HPC systems contain tens-of-thousands to hundreds-of-thousands of components. For systems of this scale, reliability becomes a major concern as the system-wide Mean-Time-Before-Failure (MTBF) decreases dramatically with the increasing count of components [38].

Recognizing the importance of system reliability, a number of methods have been presented to detect when a system fails to function properly [1]-[8]. For instance, a model-based predictive method can be applied to trigger an alert when a deviation from the derived model is observed [17],[37],[36]; several research projects have exploited data mining techniques to detect critical events in large-scale clusters by capturing and discovering fault patterns [3]. *While it is useful for system management by detecting when the system functions abnormally, it is equally important to find out which part of the system is the source of the problem*. With the root cause information in hand, system managers know where to fix the problem and application users can take corrective or preventive actions for their applications. In this paper, the process of determining the source of the problem in a large-scale cluster is regarded as *anomaly localization*.

Anomaly localization is a challenging problem, especially in large-scale systems composed of hundreds to thousands of components. Over the past decades, a number of system monitoring tools, including both open-source packages as well as commercial solutions [26]-[28], have been deployed for collecting performance and health data across the entire system. The data collection contains a wealth of information about normal and abnormal behaviors. However, due to the massive quantities of information collected from a large number of system components, the root causes of anomalies present in the data are often buried like needles in a haystack. Traditionally, human operators are responsible to check the data for possible problems by using their experience and expertise. Manual processing is time consuming and error-prone. More importantly, it does not scale. *A desirable approach is to automatically distill the data and identify the potential root causes of the problem for system managers.* Rather than working on tons of raw data collected, system managers can focus on examining these potential locations, thereby significantly reducing human efforts required for anomaly localization.

In this paper, we present an automatic method for anomaly localization in large-scale clusters, especially those used for high performance computing. Based on the observation that the nodes perform comparable activities generally exhibit similar behaviors, the proposed method explores such a similarity to find a small set of nodes that are substantially different from the majority. The result can be used as the candidate for further investigation by system managers. Our method consists of three steps: (1) *feature collection* to assemble a feature space for the system (generally has high dimensionality to capture a wide variety of system features); (2) *feature extraction* to obtain the most significant features out of the original feature space for efficient data analysis by applying the principal component analysis (PCA) algorithm [11]; and (3) *outlier detection* to quickly identify the nodes that are "far away" from the majority by using the cell-based detection algorithm [12]. PCA is used to significantly reduce time complexity for data analysis in a large data set by reducing its dimensionality, and the cell-based algorithm enables us to quickly identify the outliers in the data set. Together, these interrelated steps aim at localizing anomalies being "buried" in massive quantities of data quickly and effectively.

We have tested our method with a number of faults in a computing environment where the nodes are used for a parameter sweep simulation [24]. Under eight different fault modes, the proposed localization method is capable of discovering every fault injected, except in one case where the false negative rate is higher than zero. Besides, the false positive rate is always controlled below 0.20, meaning that less than 20% of potential

root causes are false alarms.

Different from existing studies using historical data for fault analysis [1], [4], [36], [7] (which we denote as being based on a *vertical view* of the system), our method exploits a *horizontal* view of the system for anomaly localization. Note that the proposed method is not intended to replace existing fault prediction techniques based on a vertical view, but rather to work with them. A fault prediction mechanism (e.g. a model-based or data mining based method [1]-[8]) can be used to determine when the system fails to function properly, while this work can be used to further pinpoint the potential location of the problem.

The rest of the paper is organized as follows. Section II provides a brief discussion of the related work. In Section III, we describe the anomaly localization problem, followed by a detailed description of our three-step method in Section IV. Section V presents our preliminary experiments. Section VI points out the practical usage and limitations of the proposed method. Finally, we conclude the paper in Section VII.

II. Related Work

Diagnosing system failures has been studied extensively over the past decades. Existing diagnostic techniques can be broadly classified as model-based or data mining based. A model-based approach derives a probabilistic or analytical model of the system. A warning is triggered when a deviation from the model is detected [36]. For example, Gross et al. have presented an adaptive statistical data fitting method called MSET to forecast the system dependability [37]. In [16], a naive Bayesian based algorithm is used to predict disk drive failures. In [17], a specific analytical model is developed for quickly detecting anomalies in I/O systems.

Data mining, in combination with intelligent systems, focuses on learning and classifying known faults without constructing an accurate model ahead of time. For instance, the group at the RAD laboratory applies statistical learning techniques for failure diagnosis in Internet services. In [34], Vilalta and Ma investigate frequent itemset mining for failure prediction in a networked system. In [6], Liang et al. examine several statistical based prediction techniques (e.g. spatial or temporal correlation among RAS events) for failure forecasting in a Blue Gene/L system. In our own previous works [7], we have investigated a meta-learning based method for detecting when a cluster may experience failures in the near future by adaptively combining the merits of various data-mining techniques.

Both model-based and data mining based methods mainly focus on predicting when the system behaves abnormally, while our work emphasizes on identifying where the anomaly is located, i.e. a subset of faulty nodes. Hence, our work complements existing prediction studies. Further, the majority of existing research addresses the anomaly problem by analyzing historical data. We denote such a mechanism as *a vertical approach*. Our study intends to investigate how to exploit a horizontal view of the nodes for fault diagnosis, which is considered as *a horizontal approach*. We believe both approaches should be combined and coordinated for improving fault diagnosis in large-scale systems.

The work in [18] is the most closely related work to ours. It also explores a horizontal approach for problem diagnosis. While it focuses on diagnosing application bugs, our work aims at identifying faulty nodes that may cause system failures. In addition, our work uses different techniques for data analysis. In [18], the authors apply dynamic instrumentation to collect function-level traces from each application process and then use a pairwise distance comparison to identify faulty application process(es). In our study, we investigate a three-step method, including both PCA for feature extraction and the cell-based algorithm for outlier detection.

PCA is a well-studied method, and has been applied in many fields for dimension reduction [11]. Our work is inspired by [21], [29]. Both studies use the principal component analysis method to detect anomalies in network-wide traffic. Our work differs from these studies at two aspects. First, we target localizing faulty nodes in large-scale systems such as those used for high performance computing. In doing so, our feature space is quite different from the ones used in [21], [29]. Secondly, they apply classification techniques (i.e. a supervised learning) to identify network anomalies, while we investigate the use of an optimized outlier detection method (i.e. an unsupervised learning) to find out outliers. To the best of our knowledge, we are among the first to investigate a systematic methodology for anomaly localization in HPC systems.

III. Problem Formulation

Before formulating the problem, we first describe two terms:

- *Failure*: According to the IEEE standard 610.12-1990, a failure represents the inability of a system to perform its required functions within a specified performance requirement.
- *Fault*: A fault is defined as an abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function [20].

In [19], the authors show system state transitions between faults and failures (see Figure 1). In this model, a system has three different states, namely *Good*, *Error*, and *Failed*. A fault causes the system to transit from a good state to an error state. The system in an error state generally still functions, probably under a degraded performance. The usage of the system in the error state brings the system to a failed state. In general, a latency exists between the error state and the failed state. During the latency, the system may transit back to the good state via a proper fault tolerance mechanism. Finally, the failed system returns back to the good state via some recovery mechanism.

Our aim is to troubleshoot the system in the error state by finding out the root cause of the fault (so called “anomaly”) that brings the system into an error state. Formally, the problem can be formulated as below:

Problem 1: When a system is detected in an error state (e.g. by applying model-based or data mining based techniques mentioned in Section II), the objective of anomaly localization

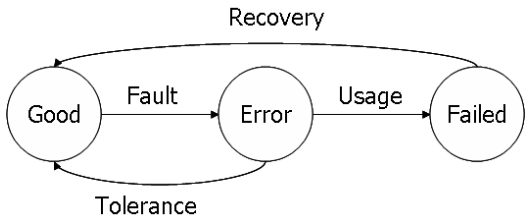

Fig. 1. System State Transitions

is to identify the nodes that cause the problem by externally observing the system behaviors.

We have made two observations on anomaly localization in large-scale clusters. First, in a cluster, the nodes performing comparable activities should exhibit similar behaviors. High performance computing clusters often have such a property, and examples include those used for parameter sweep applications [24], cluster management tools, and Web server farms [18]. Second, in most cases, the majority of the nodes are functioning normally since faults are rare events. Based on these observations, we aim at finding out a small set of nodes that exhibit different behaviors from the majority. This involves three key questions:

1) How to collect a variety of features for effectively representing node behaviors?
2) How to extract the most significant features for anomaly localization from the potentially overwhelming mass of data?
3) How to quickly identify faulty nodes?

The answers to these questions ultimately determine the coverage of anomalies and the efficiency of anomaly localization.

IV. Localization Methodology

Our proposed localization method consists of three steps: *feature collection*, *feature extraction*, and *outlier detection*. Figure 2 gives an overview of our localization methodology. The method can be triggered either periodically with a predefined frequency or by a system monitoring tool in case of unusual events.

- *Feature collection:* It is responsible for collecting an uniform set of features from the nodes and assembling the data into a matrix X. Here, a *feature* is defined as an individual measurable property of the node being observed. In this paper, a variety of features are collected from the system, including CPU, memory, I/O, and network.
- *Feature extraction:* It processes the collected data by extracting the most significant features for data analysis. A principal component analysis algorithm is applied to generate a lower dimensional matrix Y.
- *Outlier detection:* It identifies a set of nodes that are "far away" from the majority. A cell-based algorithm is used to quickly identify the outliers by using a distance metric.

TABLE I
Feature List

Features	Description
CPU_System	Percent CPU in Kernel
CPU_User	Percent CPU in User
CPU_Wait	Percent CPU blocked for I/O
Memory_Free	Amount of free memory and swap space (KB)
Page_In	Number of Pages in (KB/s)
Page_Out	Number of Pages out (KB/s)
IO_Write	Device writes per second
IO_Read	Device reads per second
IO_Svct	Average service time (ms)
Packet_In	No. of packets into the network per second
Packet_Out	No. of packets out of the network per second

A. Feature Collection

Feature collection is the prerequisite for data analysis. An anomaly typically induces changes in multiple subsystems in the node, including CPU, memory, I/O and network. For example, a memory leaking may affect the amount of free memory and the CPU utilization rate. An I/O operation to a malfunctioning disk may lead to page fault and a long CPU idle time. In order to detect a wide variety of anomalies, currently we use four system calls, namely *vmstat*, *mpstat*, *iostat* and *netstat*, to collect eleven features in the operating system layer. A short description of these features are summarized in Table I.

Let m be the number of features collected from n nodes. To capture the tendency of these features, k snapshots are sampled per node during a given time window. As shown in Figure 2, there are n matrices $X^i (i = 1, 2, \cdots, n)$, each representing the feature matrix collected from the ith node. In the matrix X^i, the element $x^i_{h,j}$ denotes the value of feature h collected at the jth snapshot, where $1 \leq j \leq k$ and $1 \leq h \leq m$.

To facilitate data analysis, we reorganize each matrix X^i into a long $(m \times k)$ column vector $x^i = [x^i_{1,1}\ x^i_{1,2} \cdots x^i_{m,k}]^T$. Together, we obtain a single large matrix:

$$X = [x^1, x^2, \cdots, x^n] \quad (1)$$

The transformation from a multiway matrix to a single large matrix makes it easy to diagnose anomalies across different nodes.

B. Feature Extraction

With the features collected from all the nodes, a simple method is to compare them for anomalies. However, such a simple comparison cannot provide an accurate result. In particular, fluctuation and noise may exist in feature values. For instance, some features could have large variances among the nodes; some features could be close to each other. Even worse, the fluctuation and noise may propagate across multiple samples, thereby impacting the accuracy of anomaly localization. *Feature selection* and *feature extraction* are two popular methods to solve the problem.

Feature selection is a process to select an optimal subset of the original features based on some criteria, such as

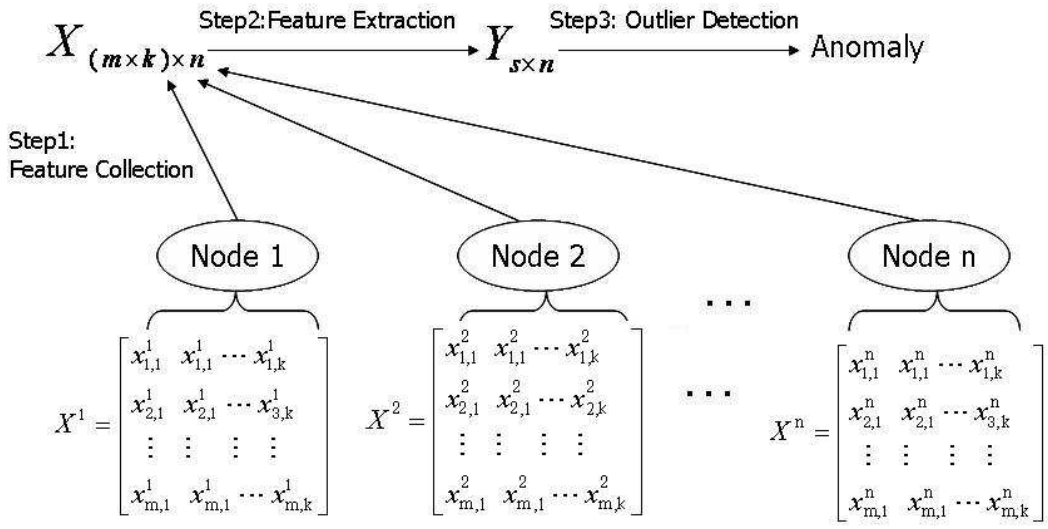

Fig. 2. Anomaly Localization. Here, m is the number of features, n is the number of nodes, and k is the number of samples collected per node

localization accuracy [11]. However, feature selection requires a training process (i.e. supervised learning). Besides, feature selection is time consuming. For instance, an optimal selection requires an exhaustive search of all the possible subsets of features. Furthermore, it generally works well with irrelevant features. As mentioned earlier, a fault may be reflected in multiple features. In other words, these features are not irrelevant, thereby greatly limiting the effectiveness of feature selection.

Different from feature selection, *feature extraction* transforms the original feature space into a space of fewer dimensions while still preserve the most significant variance in the data [11]. The data presented in a low dimensional subspace are easier to separate into different classes [21]. Further, reducing the dimensionality of the data can significantly reduce time complexity for data analysis.

In this work, we apply Principal Component Analysis (PCA) for feature extraction. PCA is a linear transformation that maps a given set of data points onto new axes (i.e. principal components) ordered by the amount of data variance that they capture [21]. One advantage of PCA is that it is unsupervised learning, meaning that we do not need to know the data label, e.g. whether it is normal or abnormal. Therefore it works well for unknown fault patterns. Another advantage of PCA is that the features in the new space are irrelevant from each other. This property ensures that the information loss caused by transforming is limited.

Our detailed steps of feature extraction are illustrated in Figure 3. Before using PCA, a *normalization* process is applied to preprocess the data. We consider that all the features are equally important. Note that the collected data may have

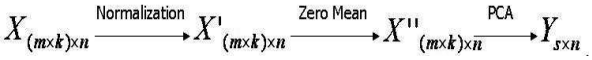

Fig. 3. Feature Extraction

different scales, e.g. memory size is generally a large number while CPU utilization is usually less than 1.0. To transform the data into a uniform scale, every data x in the matrix X is first normalized across columns. By doing so, X is normalized to X'. After this step, feature values are controlled in the range between 0.0 and 1.0. Next, the data from different nodes in the normalized matrix X' should be adjusted into *zero mean*, and the zero-meaned data is organized in the matrix X''.

The PCA method first calculates the covariance matrix of X'':

$$C = \frac{1}{n} X'' X''^T, \tag{2}$$

It then calculates the Eigenvalues of C and sorts them in a descenting order: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{m \times k}$. The first s eigenvalues that satisfy the following requirement are chosen:

$$\frac{\sum_{i=1}^{s} \lambda_i}{\sum_{i=1}^{k \times m} \lambda_i} \geq t, \tag{3}$$

where $t < 1$ is a predefined threshold.

A projection matrix $W = [w_1, w_2, \ldots, w_s]$ is determined, where w_i is the Eigenvector corresponding to the ith largest

Eigenvalues λ_i. It is also called the ith principal component. The matrix X'' is then projected into a new matrix $Y = [y_1, y_2, \ldots, y_n]$:

$$y_i = W^T x_i'' \tag{4}$$

where y_i is a s dimensional vector, $s < m \times k$.

The complexity of computing all the eigenvectors and eigenvalues of C is $O((m \times k)^3)$. Here, we only need to calculate the first s eigenvectors and eigenvalues. A neural network based method can be applied for the calculation, with a linear complexity and fast convergence [30].

C. Outlier Detection

The final step is to identify a subset of nodes that are dissimilar from the majority. In the field of data mining, these nodes are called *outliers*. Simply put, an outlier is a data point which is quite different from other data according to some criteria [12]. In this paper, we measure the dissimilarity between two data points y_a and y_b by using Euclidean distance:

$$d(y_a, y_b) = \sqrt{\sum_{i=1}^{s} (y_{a,i} - y_{b,i})^2} \tag{5}$$

where s is the dimension of the data points.

An object o in a data set T is called a $DB(p, d)$ outlier if at least fraction p of the objects in T lies greater than distance d from o. Both p and d are predefined parameters. To detect a distance-based outlier, there is no need to obtain a distribution or a probability model of the given data set, thereby making it suitable for anomaly detection in large-scale systems which generally have very complex failure modes.

A straight-forward algorithm is to calculate the number of neighbors within the distance of d for each object to determine whether it is an outlier or not. This naive algorithm, however, has the complexity of $O(sn^2)$, where s is the dimensionality and n is the number of objects in the data set. Obviously, it is not efficient, especially when there are a large number of nodes (e.g. tens of thousands to hundreds of thousands) in the system.

We use a *cell-based algorithm* for outlier detection [12]. A cell-based algorithm is an optimized outlier detection method which has a complexity of $O(c^s + n)$, where c is a small constant. Note that after the PCA processing, s is quite small (e.g. generally smaller than 5). Hence, the cell-based algorithm has a complexity that is linear with respect to n, therefore it is more time efficient as compared to a naive outlier detection algorithm.

On the new data space $Y_{s \times n}$, our cell-based algorithm works as follows. First, the data space is partitioned into cells of length $l = \frac{d}{2\sqrt{s}}$. Each cell is surrounded by two layers: (1) the first layer denoted as L_1 includes those immediate neighbors, and (2) the second layer denoted as L_2 includes those additional cells within three cells of distance. Let M be the maximum number of objects within the d-neighborhood of an outlier. Then the following three properties are the rules for outlier detection [12]:

- If there are $< M$ objects in one cell, none of the objects in this cell is an outlier.
- If there are $< M$ objects in one cell plus the L_1 layer , none of the objects in this cell is an outlier.
- If there are $\leq M$ objects in one cell plus the L_1 layer and the L_2 layer , every objects in this cell is an outlier.

The data space $Y_{s \times n}$ is then separated into two subsets: S_0 (the abnormal set) and S_1 (the normal set). For each node y_i in the abnormal set S_0, we calculate its anomaly score:

$$\eta = d(y_i, \mu), \tag{6}$$

where μ is the mean value of the elements in the normal set S_1. The anomaly score of a node reflects its fault severity; and the node with a high score has a high probability of causing the anomaly problem. Finally, the abnormal set S_0 is sorted in a descenting order of anomaly scores to system managers.

V. Preliminary Evaluation

A. Evaluation Methodology

In this section, we present our preliminary studies. Our testbed is a 47-node Sunwulf UNIX cluster at the Computer Science Department of Illinois Institute of Technique. Each node is a SUN Blade workstation 100 with one UltraSparc-IIe 500MHz CPU, 256K L2 cache and 128MB main memory. The underlying interconnect is a 100Mbps Ethernet. The system runs the SunOS 5.9 operating system. The machine is installed with SUN HPC Cluster Tools 4.0 which includes a high-performance, thread-safe, and multi-protocol implementation of the Message Passing Interface (MPI). We are currently in the process of conducting more experiments on larger scale systems at national supercomputing centers, and will present the results in our future publication.

A parameter sweep application is run on the cluster by using a master-slave model. A master node generates inputs (i.e. a 2000×2000 matrix and a 2000-dimensional vector) and sends them to 46 workers; each worker solves a set of dense linear equations by using Gaussian Elimination method [31] with the input data and then sends the results back to the master. Therefore, in this computing environment, except for the master node, all the worker nodes are performing comparable tasks, thereby expecting to exhibit similar behaviors.

We manually injected faults into the system by generating faulty threads in the background (separate from the application threads), and tested whether our localization method can effectively identify these faults. In our experiments, five types of faults were tested:

- *Memory leaking*: On a randomly selected node, besides the normal computation thread, we introduced a thread to generate memory leaking on the node, i.e. the thread continues consuming memory without releasing it periodically.
- *Unterminated CPU intensive threads*: An injected thread competes for the CPU resource with the normal computation thread on a node.

- *High frequent I/O operations*: On a randomly selected node, we introduced an I/O intensive thread which keeps reading and writing a large number of bytes from its disk.
- *Network volume overflow*: Two randomly selected nodes keep transferring a large number of packets between them.
- *Deadlock*: It is realized by blocking the process in a node for system resources.

Two accuracy metrics were used to evaluate the effectiveness of our localization method: (1) *false positive* f_p, the rate of false alarms, and (2) *false negative* f_n, the rate of missed faults. A good localization method should provide a low value (closer to 0.0) for both metrics.

The feature collection procedure was triggered every 10 seconds. On each node, eleven features (see Table I) and five samples were collected per node, which makes the matrix X of the size of $(11 \times 5) \times 46$. The threshold t in Equation 3 was set as 0.7. The parameter p and d for outlier detection were set to $p = 0.9565$ and $d = 0.4\sigma$, where σ is the variance of Y [11]. In our experiments, we found that $s = 2$ usually works well. Note that the number of snapshots and the frequency to take snapshots may influence the coverage and the effciency of anomaly localization. A detailed sensitivity study of these parameters will be part of our future work.

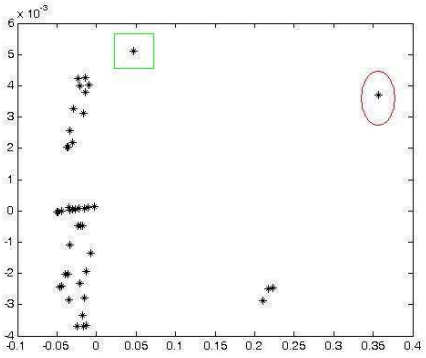

Fig. 4. Results of localizing memory leaking. Two outliers are detected, where the one in the ellipse is correct and the one in the rectangle is a false alarm.

We conducted two sets of experiments. In the first set of experiments, we randomly injected a single fault into the system. In the second set, we introduced multiple faults into the system simultaneously.

B. Results on A Single Fault

Figure 4-8 present the results by using our method on a single injected fault. On each plot, the X axis represents the $1st$ principal component, and the Y axis denotes the $2nd$ principal component. The points in the ellipse are true outliers, while those in the rectangle are false alarms.

Table II summarizes the accuracy in this set of experiments. As we can see, false negative f_n is always 0.0, meaning

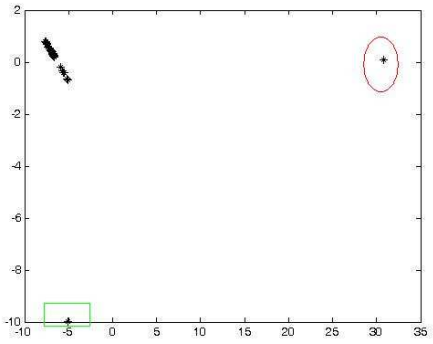

Fig. 5. Results of localizing unterminated CPU intensive threads, where the one in the ellipse is correct and the one in the rectangle is a false alarm.

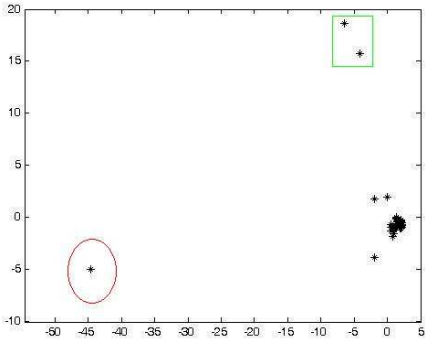

Fig. 6. Results of localizing high frequency I/O operations, where the one in the ellipse is correct and the one in the rectangle is a false alarm.

that our method is able to correctly identify all the faulty nodes. In terms of false positive, we notice an interesting phenomenon: the false positive rate is low (e.g. < 0.1) on localizing memory leak, high frequency I/O operations and deadlock, while it is relatively high (e.g. between 0.15 and 0.20) on localizing unterminated CPU intensive threads and network volume overflow. The application that we used in the experiments is CPU-intensive. As a consequence, some normal nodes which are actively engaged in computation may be erroneously identified as faulty ones. The false alarms for network volume overflow might come from fault propagation through the networks [25].

Based on the results, we then classified the faults into two groups: Type #1 with low false positive (i.e. < 0.1) and Type #2 with relatively high false positive (i.e. ≥ 0.1)). In other words, Type #1 includes memory leaking, high frequent I/O operations, and deadlock faults. Type #2 includes the faults with unterminated CPU intensive threads and network volume overflow.

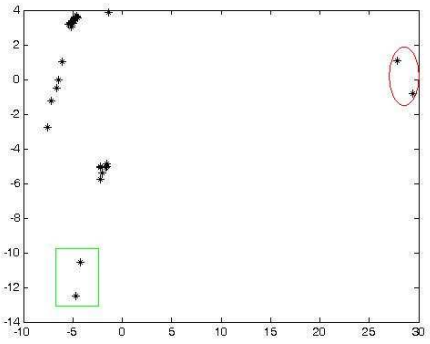

Fig. 7. Results of localizing network volume overflow, where the one in the ellipse is correct and the one in the rectangle is a false alarm.

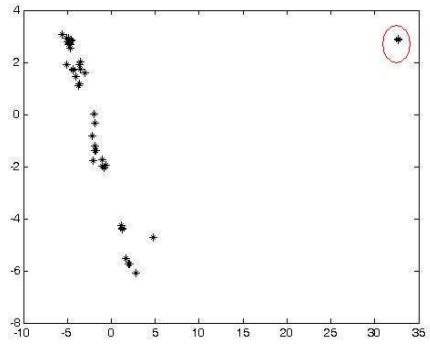

Fig. 8. Results of localizing deadlock, where the one in the ellipse is correct.

TABLE II

ACCURACY OF ANOMALY LOCALIZATION FOR THE $1st$ SET

Fault(s)	f_n	f_p
memory leak	0	0.02
unterminated CPU intensive threads	0	0.2
high frequency I/O operations	0	0.06
network volume overflow	0	0.15
deadlock	0	0.06

C. Results on Multiple Faults

In this set of experiments, we randomly injected several faults into multiple worker nodes simultaneously. Three tests were conducted: (1) injecting two Type #1 faults; (2) injecting a Type #1 fault and a Type #2 fault; and (3) injecting two Type #2 faults. The results are showed in Figure 9-11. The points in the ellipse are true outliers, those in the rectangle are false alarms, and the ones in the triangle are missed faults.

Table III lists the accuracy results for the second set of experiments. It indicates that our method is effective in localizing simultaneous Type #1 faults, as both f_n and f_p are

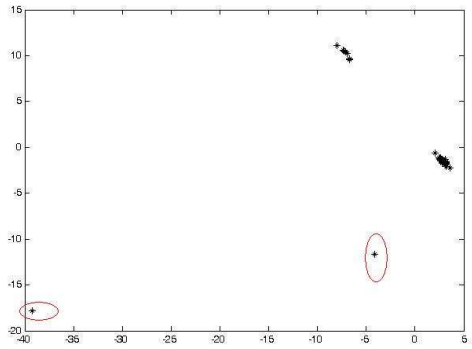

Fig. 9. Results of localizing simultaneous Type #1 faults (memory leaking and high frequent IO operation). The points in the ellipse are true outliers. The left point is from the node injected with high frequent I/O operations, and the right one is from the node injected with a memory leaking error. Both are true outliers.

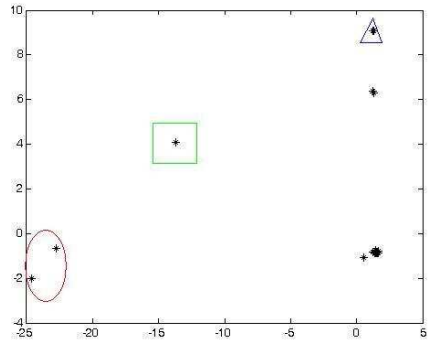

Fig. 10. Results of localizing simultaneous Type #1 and #2 faults (memory leaking and network volume overflow). The points inside of the ellipse are true outliers caused by network volume overflow, the point in the rectangle is a false alarm, and the point in the triangle is the missed fault caused by memory leaking.

TABLE III

ACCURACY OF ANOMALY LOCALIZATION FOR THE $2nd$ SET

Fault(s)	f_n	f_p
memory leak & high frequency I/O operation	0	0
memory leak & network volume flow	0.13	0.15
unterminated CPU intensive threads & network volume flow	0	0.2

zero. With mixed faults injected into the cluster, our method may fail to localize some true outliers caused by Type #1 faults. For instance, as shown in Figure 10, our method did not find the memory leaking error injected. However, we shall note that the node with memory leaking error does show different

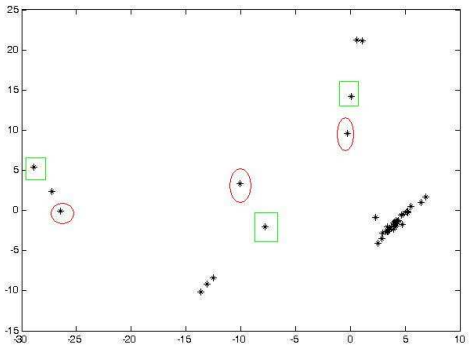

Fig. 11. Results of localizing simultaneous Type #2 faults (unterminated CPU intensive threads and network volume overflow). The identified outliers, including both true outliers and false alarms, spread across the space.

behaviors in the plot than the others. This information may be used by system managers for further investigation. Our method is able to detect simultaneous Type #2 faults, and the accuracy is similar to the case where only one Type #2 fault was injected. In summary, we conclude that mixed Type #1 and #2 faults are difficult to identified; and multiple Type #2 faults could lead to a high cost for finding the real faults.

VI. Discussion

Our method is intended to work with existing *vertical*-based detection technologies, rather than replacing them. As a working example, we can apply a data mining based method such as [7] to predict when the underlying system will experience a critical event (e.g. a hardware or software failure), and then utilize the proposed *horizontal* method to find out the potential source of the problem. Such an integration can not only tell system managers where to fix the problem, but also allow application users to take appropriate actions for fault management of their applications [10].

Similar to other automatic approaches, our localization method is also subjected to false alarms and missed detections, although the error is quite low (i.e. ≤ 0.20). This problem could be alleviated by using *supervised* methods. Nevertheless, we believe that a more promising way for anomaly localization is to involve the human in data analysis. As pointed earlier, the information produced by our localization method can be sent to a system administrator for further investigation. Other research projects have also pointed out the importance of keeping the human in failure diagnosis [32].

Our study has some limitations that remain as our future work. First, our method exploits similarity among the nodes performing comparable tasks for localizing anomalies. Hence, it cannot be *directly* applied for localizing anomalies among the nodes with wildly different behaviors. Nevertheless, a possible approach is to divide the system resources into groups of "similar" nodes and apply our method on each group. In addition, in our current design, the data analysis is performed at a central location. Such an approach cannot scale for systems with a large number of nodes. Again, a better solution is to divide-and-conquer by partitioning the system resources. We are currently working on extending this research by developing a distributed localization system.

Next, in the current work, the features are only collected in the operating system layer. We plan to include more features from both the hardware layer and the application layer. By combining a wide range of features from different computing layers, we believe it can assist in localizing faults occurring in different layers. We are also in the process of investigating other kinds of feature extraction methods, such as Independent component analysis (ICA) and Locally Linear Embedding (LLE) [33] for better localization quality.

Lastly, the paper is focused on anomalies that lie within a node, such as the faults originated from node memory, node CPU, etc. Troubleshooting anomalies occurring between nodes is a challenging problem, and there are a number of active research projects on this topic [29], [5]. In order to identify a variety of abnormal behaviors in a system, we believe it can be approached by combining different localization methods with each being applied for identifying a specific set of anomalies.

VII. Summary

We have presented an approach for anomaly localization in large-scale clusters by exploiting a horizontal view of the system. The proposed method consists of three interrelated steps: *feature collection* to acquire sufficient features to represent node behaviors, *feature extraction* via PCA to reduce the data dimensionality for efficient data analysis, and *outlier detection* via the cell-based algorithm to find out faulty nodes in a fast way. Our preliminary study has demonstrated that this localization method is capable of discovering a variety of faults, with both the false negative rate and the false positive rate controlled below 0.20.

The paper is intended to point out a potential direction for addressing the challenging problem on anomaly localization in large-scale clusters. The practical usage and the limitations of the method are also presented in the paper.

Acknowledgment

Zhiling Lan's work was supported in part by NSF grants CSR-0720549, NGS-0406328, and National Computational Science Alliance with NSF PACI Program. We would like to thank research members in the Scalable Computing Systems Laboratory at Illinois Institute of Technology for their advice and comments.

References

[1] Y. Liang, Y. Zhang, et al., "BlueGene /L Failure Analysis and Models", *Proc. of DSN06*, 2006.

[2] G. Hoffmann, F. Salfner, M. Malek, "Advanced Failure Prediction in Complex Software Systems", *Proc. of SRDS*, 2004.

[3] R. Sahoo, A. Oliner, et al., "Critical Event Prediction for Proactive Management in Large-scale Computer Clusters", *Proc. of KDD 2003*,pp. 426-435,2003.

[4] K. Trivedi and K. Vaidyanathan, "A Measurement-based Model for Estimation of Resource Exhaustion in Operational Software Systems", *Proc. of the 10th International Symposium on Software Reliability Engineering*, 1999.

[5] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, E. Brewer , "Failure Diagnosis Using Decision Trees", *International Conference on Autonomic Computing (ICAC-04)*, 2004.

[6] R. Vilalta and S. Ma, "Predicting Rare Events in Temporal Domains", *Proc. of IEEE ICDM*, 2002.

[7] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A meta-learning failure predictor for bluegene/l systems," *Proc. of ICPP'07*, 2007.

[8] J. Brevik, D. Nurmi, and R. Wolski, "Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-Peer Systems", *Proc. of IEEE CCGrid*, 2004.

[9] J. Zhang and R. Figueiredo, "Application classification through monitoring and learning of resource consumption patterns", *10th IEEE International Paralle & Distributed Processing Symposium*, Rhodes Island, Greece, Apr. 25-29, 2006.

[10] Yawei Li and Zhiling Lan, "Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster Computing", *Proc. of IEEE CCGrid'06*, 2006.

[11] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley-Interscience, New York, NY, 2001. 2nd edition.

[12] Edwin M. Knorr, Raymond T. Ng, Vladimir Tucakov, "Distance-based outliers: algorithms and applications", *The VLDB Journal*,(2000) 8: 237-253.

[13] The TOP500 Supercomputer Sites. http://www.top500.org

[14] Hardware monitoring by lm sensors http://secure.netroedge.com/-lm78/info.html

[15] B. Allen, "Monitoring Hard Disk with SMART", *Linux Journal*, January, 2004.

[16] Greg Hamerly and Charles Elkan, "Bayesian approaches to failure prediction for disk drives", *ICML 2001*,pp. 1-9, 2001.

[17] Kai Shen, Ming Zhong, and Chuanpeng Li, "I/O System Performance Debugging Using Model-driven Anomaly Characterization", *4th USENIX Conference on File and Storage Technologies*, pp. 309C322, 2005.

[18] A. V. Mirgorodskiy, N. Maruyama, B.P. Miller, "Problem Diagnosis in Large-Scale Computing Environments", *Supercomputing, 2006*, pp. 11-24, 2006.

[19] R. Chillarege and N. S. Bowen, "Understanding large system failure–A fault injection experiment", *Proc. 19th Int. Symp. Fault-Tolerant Comput.*, pp. 355-363, June 1989.

[20] Nancy Leveson, "Safeware: System safety and computers", *New York: Addison-Wesley*, 1995.

[21] A. Lakhina, M. Crovella, C. Diot, "Mining anomalies using traffic feature distributions",*Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications* , 2005.

[22] K. Vaidyanathan, K. S. Trivedi, "A measurement-based model for estimation of resource exhaustion in operational software systems", *Proceedings of the 10th International Symposium on Software Reliability Engineering*, pp.84-93, 1999.

[23] B. Schroeder, G. A. Gibson , "A Large-scale Study of Failures in High-performance-computing Systems", *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 249-258, 2006.

[24] H. Casanova, G. Obertelli, F. Berman and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid", *Supercomputing, 2000*, pp. 60-69, 2000.

[25] Girard, A. , Brunilde, S., "Multicommodity flow models, failure propagation, and reliable loss network design", *IEEE/ACM Transactions on Networking (TON)*, Volume 6, Issue 1, pp. 82-93, 1998.

[26] Ganglia Monitoring System. http://ganglia.sourceforge.net/

[27] M. Scottile and R. Minnich, "Supermon: A High- Speed Cluster Monitoring System", *Proc. IEEE Cluster*, 2002.

[28] S. Smallen, C. Olschanowski, K. Ericson, P. Bechman, and J. Schopf, "The Inca test harness and reporting Framework", Proc. of SC04, 2004.

[29] A. Lakhina, M. Crovella, C. Diot, "Diagnosing Network-Wide Traffic Anomalies", *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications* , 2004.

[30] Rao. Y. N. and Principe. J. C., "A fast, On-Line Algorithm for PCA and Its Convergence Characteristics", *Proceedings of the 2000 IEEE Signal Processing Society Workshop*, vol.1, pp. 299-307, 2000.

[31] Ananth Grama, Vipin Kumar, Anshul Gupta, George Karpis, "Introduction to Parallel Computing (2nd Edition)", *Addison-Wesley*, 2003.

[32] P. Bodik, G. Friedman, L. Biewald, H. Levine, G. Candea, K. Patel, G. Tolle, J. Hui, A. Fox, M. I. Jordan and D. Patterson, "Combining Visualization and Statistical Analysis to Improve Operator Confidence and Efficiency for Failure Detection and Localization", *The 2nd IEEE International Conference on Autonomic Computing (ICAC '05)*, 2005.

[33] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. Science, 2000, Vol. 290: 2323-2326

[34] R. Vilalta and S. Ma, "Predicting rare events in temporal domains," in *Proc. of IEEE Intl. Conf. On Data Mining*, 2002.

[35] G. Hamerly and C. Elkan, "Bayesian approaches to failure prediction for disk drives," in *Proc. of ICML*, 2001.

[36] J. Hellerstein, F. Zhang, and P. Shahabuddin, "A statistical approach to predictive detection," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 2001.

[37] K. Vaidyanathan and K. Gross, "MSET Performance Optimization for Detection of Softtware Aging", *Proc. of ISSRE*, 2003.

[38] J. Sancho, F. Petrini, G. Johnson, J. Fernandez, and E. Frachtenberg, "On the feasibility of incremental checkpointing for scientific computing," in *Proc. of IPDPS'04*, 2004.