A Scalable, Non-Parametric Method for Detecting Performance Anomaly in Large Scale Computing

Li Yu, *Student Member, IEEE* and Zhiling Lan, *Senior Member, IEEE*

Abstract—As computer systems continue to grow in scale and complexity, performance problems become common and a major concern for large-scale computing. Performance anomalies caused by application bugs, hardware or software faults, or resource contention can have great impact on system-wide performance and could lead to significant economic losses for service providers. While many detection methods have been presented in the past, the newly emerging challenges are detection scalability and practical use. In this paper, we propose a scalable, non-parametric method for effectively detecting performance anomalies in large-scale systems. The design is generic for anomaly detection in a variety of parallel and distributed systems exhibiting peer-comparable property. It adopts a divide-and-conquer approach to address the scalability challenge and explores the use of non-parametric clustering and two-phase majority voting to improve detection flexibility and accuracy. We derive probabilistic models to quantitatively evaluate our decentralized design. Experiments with a suite of applications on production systems demonstrate that this method outperforms existing methods in terms of detection accuracy with a negligible runtime overhead.

Index Terms—Large-scale systems, performance anomalies, hierarchical grouping, non-parametric clustering

1 Introduction

Performance anomalies, which may be caused by various factors such as resource contention, application bugs, and hardware/software faults, become common and a major concern for large-scale computing. A performance anomaly is a type of error, indicating a deviation from the expected performance of a computing node or device. While performance anomaly usually appears only on a portion of the system and does not necessarily lead to a compete crash of the computing, it can introduce serious performance impacts (e.g., greatly slowing down task execution). Studies have shown that a single problematic node can lead to job completion time being extended up to five times [1]. Consequently, an effective anomaly detection method is crucial for large scale computing not only to mitigate system-wide performance degradation, but also to improve cost efficiency for service providers.

Commonly adopted detection approaches can be generally categorized as either model-based or data-driven. A model-based approach derives a probabilistic or analytical model of the system and uses it as a reference for evaluation [2], [3], [4], [5], [6], [7]. Despite its broad usability, a major limitation of model-based approach is the difficulty of generating and maintaining a consistent model for systems with high variability.

A data-driven approach, in contrast, relies only on system data for decision making, thus being applicable to a variety of dynamic computing environments. For anomaly detection in large scale computing, the data-driven approach based on node comparison has been widely studied [8], [9], [10], [11], [12], [13]. These methods assume a peer-comparable environment where nodes performing comparable activities are expected to exhibit similar behaviors, differing in ways they use to compare node similarity.

As computer systems continue to grow in scale and complexity, comparison-based detection methods faces two key challenges. First is *scalability*. Existing methods are typically based on a centralized design, where a central node is responsible for collecting data from other nodes and analyzing them together for decision making [8], [9], [10]. Although these methods provide good detection accuracy, they fail to meet the scalability requirement. A large-scale system may consist of thousands or hundreds of thousands of nodes. The computation cost required to analyze the huge volume of data at the central node is non-trivial. Moreover, the communication cost involved to transfer data from all other nodes to the central node can be significant, especially when the computing resources are remotely located.

Second is *practical use*. Existing detection schemes tend to use parametric methods that require a manual parameter tuning process based on a prior knowledge of system characteristics. For example, some methods need to adjust threshold values to achieve optimal detection accuracy according to a predefined system anomaly rate [8], [14], [15]; some others assume the number of abnormal behavior types is known in advance [11], [12], [13]. In practice, these assumptions hardly hold due to the fact that in large-scale systems with high complexity, anomaly rate can vary and problematic nodes can behave distinctly due to different

• *The authors are with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616.*
E-mail: lyu17@hawk.iit.edu, lan@iit.edu.

Manuscript received 13 Oct. 2014; revised 25 Aug. 2015; accepted 25 Aug. 2015. Date of publication 1 Sept. 2015; date of current version 15 June 2016.
Recommended for acceptance by R. Brightwell.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2475741

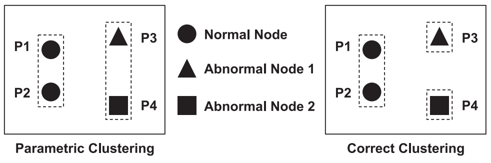


Fig. 1. Limitation of parametric detection methods. P3 and P4 are abnormal but exhibit distinct behaviors due to different root causes. (left) A parametric clustering method arbitrarily categorizes the nodes into two groups, thus failing to identify the anomalies by using a majority voting based detection mechanism. (right) An ideal method should divide these nodes into three groups.

root causes. Fig. 1 illustrates an example where a parametric method fails to identify the anomaly.

To address the above challenges, the primary contribution of this paper lies in a collection of techniques. First, we adopt *a grouping strategy*, through which we divide the big problem involving the analysis of a large system into many small problems while at the same time maintain the peer-comparable environment. As the number of nodes in each group is limited and only local communication is needed for problem solving, such a decentralized design is able to achieve high scalability. Second, we explore *a non-parametric clustering method* that does not rely on any pre-defined cluster numbers and thus is capable of handling multiple anomaly types such as the example shown in Fig. 1. Third, we develop *a two-phase majority voting mechanism* to improve anomaly detection in case of high anomaly rate. We also derive *probabilistic models* to quantify detection accuracy of our decentralized design under various configurations. Together, these techniques form a scalable and non-parametric detection framework for large-scale computing.

We demonstrate the effectiveness of our design by means of a set of experiments on two systems. One is a production system named *Stampede* at Texas Advanced Computing Center (TACC). It consists of more than $6,000$ compute nodes and is mainly used for large scale scientific computing [16]. The other is a local cluster with 64 compute nodes, which is mainly used for data intensive computing. We inject a variety of anomaly types into the computing, and test whether the proposed method is capable of pinpointing these anomalies accurately under various computing scales. Experiments clearly demonstrate that our decentralized design is highly scalable and outperforms existing detection schemes by 12 percent on average in terms of detection accuracy. The proposed detection method incurs a very low runtime overhead, e.g., less than 120 ms for performance anomaly detection of 2K nodes on Stampede.

The rest of the paper is organized as follows. Section 2 describes basic assumptions of this study. Section 3 gives the details of methodology. Section 4 present analytical models to quantify method accuracy. Sections 5 and 6 present experiments and list experimental results. Section 7 discusses limitations and the potential use of the proposed method, followed by related work in Section 8. Finally, this work is concluded in Section 9.

2 Problem Statement

In large-scale computing, performance anomalies are an important class of errors that are difficult to diagnose and isolate. Performance anomalies can stem from different system layers such as a bug in the application code, a flaw in the hardware or software, an unexpected interaction between computing components, or a conflict over access to a shared resource such as network devices. They often result in a "limping-but-alive" system, i.e., the system continues to work, but with degraded performance. Google reported that more than 95 percent machines experience a variety of performance problems like packet-loss, disk hogs, random connectivity lost, etc. in the first year of a cluster's operation [17]. Studies on system logs showed that more than 30 percent of Hadoop bugs are manifested as degraded performance instead of system crash [18].

Performance anomalies can cause great impact on system performance. A single poorly performing node can significantly affect the behavior of the overall system and introduce system-wide performance issues like poor task turnaround time, low job response time, system throughput drop, and violation of service level agreement. These performance issues can lead to poor system utilization of expensive high-performance computing systems as well as economical losses for business service providers.

Paper goals and non-goals. In this work, we seek to develop a decentralized method for detecting performance anomalies in large-scale computing. The goal is to identify the computing nodes that undergo performance problems (not necessarily leading to system crash) and cause an application to take longer time to complete than that if the problem were not present. The detection is based on both OS level (black-box) performance metrics and middleware level (white-box) performance metrics. In addition, an important feature of this work is application-transparent, meaning that our method does not require any modifications of the hosted applications. We shall point out that in this work, we do not target fail-stop failures nor code-level debugging. Further, this work is concentrated in detecting node-level performance anomalies, and subsequent correction operations after detection are beyond the scope of this paper. We believe various methods such as process migration [19] and/or those listed in [20] can be taken for error handling.

Assumptions. Our design is based on two assumptions. First is the peer-comparable property. In a peer-comparable environment, nodes performing comparable activities are expected to exhibit similar behaviors. Peer comparability is commonly observed in parallel computing environments. For instance, in the MPI paradigm, when an application is well distributed across multiple compute nodes, these computing nodes often show similar patterns and complete their tasks around the same time. Distributed computing like Map/Reduce may also form a peer-comparable environment in case of homogeneous hardware resources. Furthermore, this assumption can be relaxed for peer comparability per group, rather than across the entire system. As we will show in Section 3.1, the grouping method can be applied to provide a peer-comparable environment in each group. Second, we assume that normal nodes are the majority and have similar behaviors. This is a minor requirement given that anomalies are rare events. Both assumptions are commonly used for any detection methods based on similarity comparison [21].

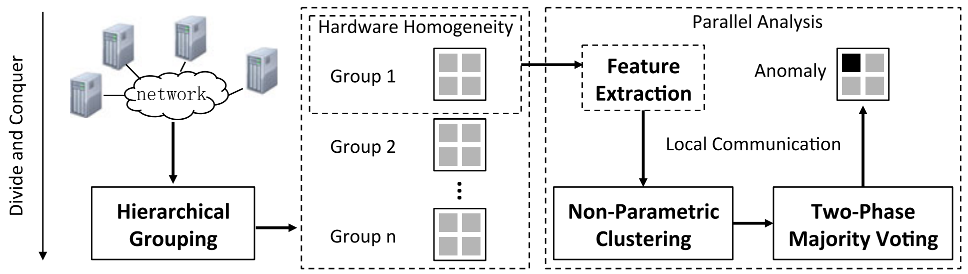


Fig. 2. Overview of our anomaly detection framework.

3 Methodology

Fig. 2 depicts our design and its main components, including hierarchical grouping, feature extraction, non-parametric clustering and two-phase majority voting.

3.1 Hierarchical Grouping

Grouping denotes a process of splitting a group of system nodes into a number of smaller subgroups [22], [23]. The purpose of hierarchical grouping is to avoid global computation and communication for decision making, and to guarantee a peer-comparable environment within each group. Specifically, our grouping strategy is conducted in three steps:

- *Geographical grouping*. Computing nodes are grouped according to their geographical locations.
- *Topology-aware grouping*. Computing nodes are further divided based on their network topologies and hardware configurations.
- *Random ouping*. Finally, every node serves as a central node and forms a group by randomly assigning n neighbors to it.

Geographical grouping is conducted first, which is applied to avoid the long distance communication between remote nodes. The goal of topology-aware grouping is to further reduce the group size and maintain hardware homogeneity. The rule in this step varies according to different system environments. For example, compute nodes built on a fat-tree topology can be grouped by switches in the network. The last step is adopted when the group size is still large after the first two grouping steps, e.g., a group contains hundreds of identical nodes that are fully connected via a local area network. Using the random grouping strategy, a node may belong to one or more groups, but its state is only determined by the group where it is the central node.

With respect to the third step, we note that other grouping strategies like nearest neighbor (NN) based grouping or grouping based on manager/worker thread similarity could be used. While these grouping strategies could bring additional benefits (e.g., reducing workload dependent variability in the local group), they run counter to the application-transparent goal of our detection design. An important reason of choosing random grouping is that it can decrease the possibility of our method falling into a local trap, which happens when a performance anomaly propagate to its nearest neighbors and make the majority of a local group become abnormal.

As nodes only communicate with neighbors within the same group, the group size is an important factor for both detection accuracy and detection overheads. A smaller group size indicates lower communication cost and faster detection, but suffers from low detection accuracy due to insufficient number of comparable samples (e.g., abnormal nodes make up the majority of the group). A larger group size usually provides better detection accuracy; however, it introduces more overheads due to higher computation and communication cost. Hence, an important task of this work is to strike for a balance between accuracy and speed. We will present an analytical study to quantify the tradeoff in Section 4.

3.2 Feature Extraction

Following hierarchical grouping, our design performs multiple group analysis concurrently. In each group, we collect data to characterize node behaviors, and transfer them into a uniform format for further analysis. The data gathered from each group are put into a $m * n$ matrix X, where m is the number of features (rows) per node and n is the number of nodes (columns) in the group. The value of m can be further represented as $m = c * t$, where c is the number of features gathered to characterize node behaviors and t is the number of snapshots sampled per node. As the collected data have different scales, the matrix X is normalized across columns such that all feature values fall into the range of 0.0 and 1.0.

Next we project high-dimensional data to a space with lower dimensionality using feature extraction. Feature extraction brings several benefits to anomaly detection. First, it captures useful information hidden in features that is hard to discover via direct comparison. Second, it removes noise in data, thus improving detection accuracy. Third, a lower feature space can reduce diagnosis overhead significantly. Feature extraction techniques like principal component analysis (PCA) [24] and independent component analysis (ICA) [25] have been adopted in a number of anomaly detection studies [26], [27], [28]. In this study, we apply kernel principal component analysis (KPCA), a non-linear extension of PCA, to project the original feature space to a three-dimensional feature space. The main reason of using KPCA, rather than PCA, is that KPCA has proven to provide better recognition rate than PCA [29]. According to the literature, a two-dimensional or three-dimensional projected feature space is sufficient to provide good detection accuracy.

3.3 Non-Parametric Clustering

Clustering analysis is used to distinguish node behaviors within the same group. Commonly adopted clustering methods can be categorized into four types, including centroid-based (e.g., k-means), connectivity-based (e.g.,

TABLE 1
Notations for the Two-Phase Voting Method

n	group size
p	anomaly probability of a node
$\mathbf{a}_i$	the central node in a group
S_i	state of $\mathbf{a}_i$, $S_i \in \{M, F, N, A\}$, where M = Majority, F = Fewness, N = Normal and A = Abnormal
$M(\mathbf{a}_i)$	set of group members of $\mathbf{a}_i$ being labeled with M after the first phase
$F(\mathbf{a}_i)$	set of group members of $\mathbf{a}_i$ being labeled with F after the first phase
$G(\mathbf{a}_i)$	set of group members of $\mathbf{a}_i$, where $G(\mathbf{a}_i) = M(\mathbf{a}_i) \bigcup F(\mathbf{a}_i)$ and $\lvert G(\mathbf{a}_i)\rvert = n$
$C(\mathbf{a}_i)$	set of group members of $\mathbf{a}_i$ belonging to the same cluster as $\mathbf{a}_i$
$P_{S\rightarrow\tilde{S}}$	probability of $\mathbf{a}_i$ with true state S being labeled with state $\tilde{S}$

hierarchical), distribution-based (e.g., Gaussian mixture) and density-based (e.g., DBSCAN and mean-shift) [30], [31], [32]. Despite different types, most clustering algorithms require data information as input parameters. For example, k-means and hierarchical clustering need the number of clusters, DBSCAN needs the neighborhood size and mean-shift needs a predefined bandwidth value for the searching window. However, as stated earlier, the assumption of priori knowledge can cause serious detection issues in practice.

In this study, we explore a density-based clustering algorithm called *adaptive mean shift clustering* (AMS) that requires no priori knowledge of the data [33]. Compared to commonly used parametric methods, AMS makes fewer assumptions about system characteristics. It differs from mean-shift clustering in that its searching bandwidth can be determined automatically. Specifically, for each data point $\mathbf{x}_i (i = 1, \ldots, n)$ that is associated with a bandwidth h_i, AMS involves following steps:

1) A window is created with a bandwidth h_i for each $\mathbf{x}_i$.
2) The mean of the data in the window is calculated and denoted as the new center $\tilde{\mathbf{x}}_i$.
3) Starting from $\tilde{\mathbf{x}}_i$, step 2 and 3 are repeated until a convergence.
4) Finally, data points leading to the same center are assigned to a cluster.

The new center in Step 2 is calculated by:

$$\tilde{\mathbf{x}}_i = \frac{\sum_{j=1}^{n} \mathbf{x}_{i,j} g(\| \frac{\mathbf{x}_i - \mathbf{x}_{i,j}}{h_i} \|^2)}{\sum_{j=1}^{n} g(\| \frac{\mathbf{x}_i - \mathbf{x}_{i,j}}{h_i} \|^2)}, \tag{1}$$

where each $\mathbf{x}_{i,j}$ represents a point in the search window of $\mathbf{x}_i$ and $g(x) = -k'(x)$. The function $k(x), 0 \leq x \leq 1$, is called the profile of the kernel and is used to estimate the density at location $\mathbf{x}_i$ in the feature space. The bandwidth h_i is calculated in an adaptive manner and defined as a $L1$ norm:

$$h_i = \|\mathbf{x}_i - \mathbf{x}_{i,n/2}\|_1, \tag{2}$$

where $\mathbf{x}_{i,n/2}$ is the $n/2$ nearest neighbor of the point $\mathbf{x}_i$ and n is the number of nodes in the group.

The time complexity of AMS is $O(Tn^2)$, where T is the number of iterations. Due to the small number of nodes in each group and the low dimension feature space after data transformation, the algorithm converges very fast (typically within several milliseconds) in our experiments, indicating a negligible overhead.

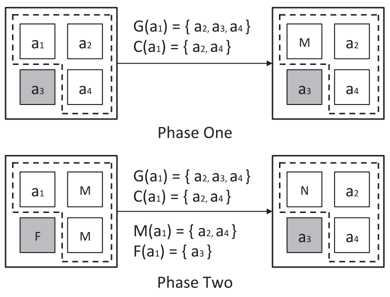


Fig. 3. In this example, there are two clusters $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_4\}$ and $\{\mathbf{a}_3\}$ in the group where $\mathbf{a}_1$ is the central node. In phase one, $|G(\mathbf{a}_1)|$ = 3 and $|C(\mathbf{a}_1)|$ = 2, thus $\mathbf{a}_1$ is labeled with M. In phase two, assuming $\mathbf{a}_2$ and $\mathbf{a}_4$ are labeled with M and $\mathbf{a}_3$ is labeled with F after the first phase, then $|C(\mathbf{a}_1) \cap M(\mathbf{a}_1)|$ = 2 and $|M(\mathbf{a}_1)|$ = 2, thus $\mathbf{a}_1$ is finally labeled with N.

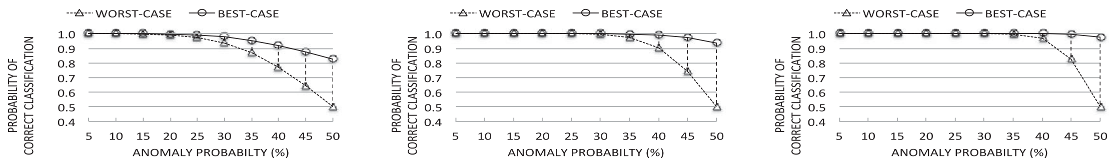
3.4 Two-Phase Majority Voting

Based on the clustering, the next component of our design is two-phase majority voting, aiming to identify abnormal nodes in each group. Majority voting was introduced by Blough et al. to detect faulty processors in multiprocessor systems [34]. Several variants have been applied to fault detection in wireless network [35], [36]. In this study, we extend the basic majority voting algorithm to a two-phase version for better detection accuracy. In the first phase, a node is labeled with M ("Majority") if it belongs to the majority of all group members; otherwise it is labeled with F ("Fewness"). In the second phase, only the nodes labeled with M have the right to vote. A node is labeled with N ("Normal") if it belongs to the majority of the group members labeled with M; otherwise, it is labeled with A ("Abnormal").

Algorithm 1. Two-phase Majority Voting

```
 1: Given $G(\mathbf{a}_i)$ and $C(\mathbf{a}_i)$
 2: if $|C(\mathbf{a}_i)| > |G(\mathbf{a}_i)|/2$ then
 3:     $S_i \leftarrow M$
 4: else
 5:     $S_i \leftarrow F$
 6: end if (Phase one ends here)
 7: if $M(\mathbf{a}_i) \neq \emptyset$ then
 8:     if $|C(\mathbf{a}_i) \cap M(\mathbf{a}_i)| \geq |M(\mathbf{a}_i)|/2$ then
 9:         $S_i \leftarrow N$
10:     else
11:         $S_i \leftarrow A$
12:     end if
13: else
14:     if $S_i = M$ then
15:         $S_i \leftarrow N$
16:     else
17:         $S_i \leftarrow A$
18:     end if
19: end if (Phase two ends here)
```

Algorithm 1 presents our two-phase voting mechanism using notations summarized in Table 1. All the central nodes of the groups execute the algorithm concurrently. At the beginning of Phase two, every central node needs to communicate with its neighbors to get their labels obtained from Phase one. Fig. 3 gives an example of the



(a) n=4 (b) n=8 (c) n=12

Fig. 4. Detection accuracy of the proposed method under different group sizes and anomaly probabilities. The gap between the *best-case* (solid line) and *worst-case* (dash line) is given by Equation (5).

two-phase majority voting process in a group, where $\mathbf{a}_1$ is the central node.

3.5 Implementation

The implementation of our design includes both an online version and an offline version. For the online version, we use C programming language based on MPI paradigm. MPI group routines such as MPI_Comm_group and MPI_Group_incl are adopted to implement our grouping strategy; MPI_Gather function is used to collect sample data within each group. The offline version is based on MATLAB, which requires log data (i.e., node metrics collected from the system) as input and is mainly used for the comparison of our method with existing anomaly detection methods. The downloads are available at *http://bluesky.cs.iit.edu/snd/*.

4 Analytical Analysis

In this section, we derive probabilistic models to quantify detection accuracy of our design under different configurations. These models enable us to analyze the impacts of different factors on detection accuracy. The factors include group size, anomaly probability, number of anomaly types and number of voting phases.

Using the notations listed in Table 1, the probability of a normal node being labeled with M in Phase one is given by Equation (3), where f is the number of abnormal nodes in the group,

$$sP_{N\to M} = (1-p)\sum_{f=0}^{\lceil \frac{n}{2}\rceil -1} C_n^f (1-p)^{n-f} p^f. \tag{3}$$

The probability of a normal node being labeled with F in Phase one can be given in a similar way. It is also defined as:

$$P_{N\to F} = 1 - p - P_{N\to M}. \tag{4}$$

The probability of an abnormal node being labeled with F in Phase one is given by Equation (5). In the *best-case*, all the abnormal nodes behave differently and are labeled with F in Phase one. In the *worst-case*, all the abnormal nodes behave similarly (e.g., caused by the same root cause) and are put into the same cluster,

$$P_{A\to F} = \begin{cases} p & best-case \\ p\sum_{f=0}^{\lfloor \frac{n}{2}\rfloor} C_n^f (1-p)^{n-f} p^f & worst-case \\ \quad .. & \end{cases} \tag{5}$$

The probability of an abnormal node being labeled with M in Phase one can be given similarly.

$$P_{A\to M} = p - P_{A\to F}. \tag{6}$$

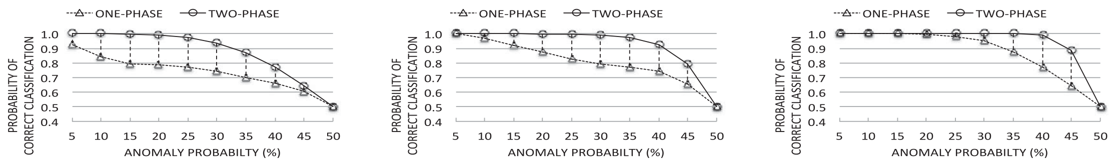
Based on Equations (3)-(6), we obtain the probability of a normal node being correctly labeled as below,

$$\begin{aligned} P_{N\to N} = (1-p)\sum_{m=1}^{n} C_n^m \left(\sum_{a=0}^{\lceil \frac{m}{2}\rceil -1} C_m^a P_{N\to M}^{m-a} P_{A\to M}^a\right) \\ \left(\sum_{b=0}^{n-m} C_{n-m}^b P_{N\to F}^b P_{A\to F}^{n-m-b}\right) + P_{N\to M}\left(\sum_{c=0}^{n} C_n^c P_{N\to F}^c P_{A\to F}^{n-c}\right). \end{aligned} \tag{7}$$

Similarly, we can derive $P_{A\to A}$. The detection accuracy of our method is calculated as $P_{N\to N} + P_{A\to A}$, indicating the probability of a node being correctly classified.

These probabilistic models allow us to analytically evaluate our design under a variety of configurations. We can examine detection accuracy of our method under different group sizes and anomaly probabilities, and the results are presented in Fig. 4. It shows that detection accuracy increases as group size grows, but the improvement becomes not obvious when the group size is larger than eight. Also, accuracy gap exists between the *best-case* and *worst-case* given by Equation (5), which is particularly significant when anomaly probability is high (i.e., $p \geq 20\%$). This observation implies that if our method is able to distinguish different abnormal behaviors, the accuracy of the voting mechanism can be improved. In other words, in case of high anomaly probability, the non-parametric clustering can greatly outperform parametric clustering methods that assume a predefined number of behavioral patterns.

We can also use these models to compare our two-phase voting with the conventional one-phase voting, and the results are presented in Fig. 5. It clearly demonstrates that the two-phase majority voting outperforms one-phase



(a) n=4 (b) n=8 (c) n=12

Fig. 5. Comparison of our two-phase majority voting represented by solid curves and the conventional one-phase majority voting represented by dash curves.

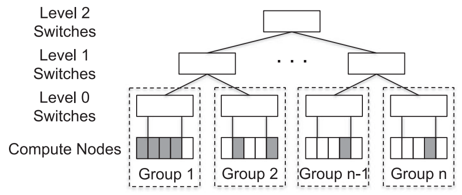


Fig. 6. The network topology of stampede.

voting. According to the probabilistic models, two-phase voting gives significantly better detection accuracy than the one-phase version until the anomaly probability reaches 50 percent. Actually, we can extend the two-phase voting to n-phase voting by repeating line 7 to line 19 of Algorithm 1 in a similar manner, i.e., only the nodes labeled as majority in last phase have the right to vote in the current phase. However, according to our study, n-phase voting ($n > 2$) can not deliver significantly better accuracy than two-phase voting, but rather introduce higher overhead (i.e., for n-phase voting, we need n times communication among nodes within each group). Two-phase voting provides a good tradeoff between detection accuracy and detection overhead.

5 Experiment Configuration

5.1 Testbeds

5.1.1 Stampede at TACC

Stampede is one of the most powerful supercomputers in the world [37]. It is a 6,400-node system located at TACC. Each node is configured with two Intel Xeon E5-2680 processors, 32 GB memory and a 250 GB local disk. All nodes are interconnected with Mellanox FDR, the InfiniBand technology in a two-level fat-tree topology. The network topology of Stampede, along with our grouping, is described in Fig. 6.

Following the three steps listed in Section 3.1, we apply our hierarchical grouping strategy to Stampede by leveraging its topology query service [38]. In the first step, all compute nodes are put into one group because they are assembled in the same machine room. In the second step, we divide the compute nodes into groups according to the level 0 switches as shown in Fig. 6. After this step, the distance between nodes within the same group is two hops. In the third step, we further divide nodes by randomly assigning each node with $n - 1$ neighbors, where n is the group size for our parallel analysis.

We evaluate our design with five application kernels and two proxy applications. The five kernels, summarized in Table 2, are from a spectrum of computational domains, including sparse linear algebra, N-body methods, structured grids, spectral methods and Monte Carlo [39], [40], [41]. The two proxy applications include a multi-physics multi-scale simulation code and an open-source computational fluid dynamics solver [42], [43].

5.1.2 HEC Cluster

HEC is a 65-node computing system located at Illinois Institute of Technology. It consists of 64 compute nodes and one head node. Each computing node has two Quad-Core AMD Opteron(tm) processors, 8 GB memory and a 250 GB 7200 RPM SATA-II disk. All the nodes are equipped with Gigabit Ethernet interconnection. Similarly, in the first step of hierarchical grouping, we put all compute nodes into one group as they are located within the same cluster. In the second step, we divide the nodes into three groups according to their physical distances, which are determined by the three switches in the network. In the third step, we further divide nodes within each group using our random grouping strategy.

Hadoop benchmark *TeraSort*, *Bayesian Classification* and *Hive Join* server as the test workloads on HEC [44]. These three workloads are summarized in Table 2 as well.

5.2 Fault Injection

We randomly inject multiple faults into the testbeds by generating faulty threads in the background, separating from the application threads. Three factors are considered in our fault injection. First is the number of nodes to inject faults, second is the location of nodes to inject faults, and the last is the types of faults to inject. In our experiments, the number of nodes to inject faults is determined by anomaly probability. For example, given 100 nodes and 10 percent anomaly probability, we randomly inject faults into 10 nodes out of the 100 nodes. In addition, we inject different number of fault types, denoted as *one-anomaly* to *four-anomaly*, and evaluate them separately. The injected fault types are summarized as below.

- *CPU-intensive threads:* On randomly selected nodes, the injected threads compete for the CPU resource with the normal computation on the nodes.

TABLE 2
Workloads Used in this Work

Workload Name	Category	Programming Model	Testbed	Implementation	Input size
Conjugate Gradient (CG)	Sparse linear algebra	MPI	Stampede	NPB CG [39]	Class C
Barnes-Hut simulation (NB)	N-body method	MPI	Stampede	[40]	10,000 Particles
Multi-grid (MG)	Structured grids	MPI	Stampede	NPB MG [39]	Class C
FFT (FT)	Spectral methods	MPI	Stampede	NPB FT [39]	Class C
Monte Carlo simulation (MC)	Monte Carlo	MPI	Stampede	XSBench [41]	Large, Lookups = 10^8
Flash (FL)	Multi-physics simulation	MPI	Stampede	FLASH [42]	maxPerProc = 10^5
Nek5000 (NK)	Computational fluid dynamics	MPI	Stampede	Nek5000 [43]	Large
TeraSort (TS)	Sorting	Map/Reduce	HEC	[44]	10 GB data / node
Bayesian Classification (BC)	Machine Learning	Map/Reduce	HEC	[44]	5 GB data / node
Hive Join (HJ)	Analytical Query	Map/Reduce	HEC	[44]	5 GB data / node

TABLE 3
The Summary of Collected Features

	Name	Description
OS Layer	CPU_SYS	CPU utilizations
	CPU_USER	
	CPU_WAIT	
	MEMORY_Free	Free memory (KB)
	MEMORY_Swapped	Used virtual memory (KB)
	DEV_BLOCK_IN	No. of blocks sent / s
	DEV_BLOCK_OUT	No. of blocks read / s
	PACKET_IN	No. of packets received / s
	PACKET_OUT	No. of packets sent / s
	NET_BYTE_IN	Bytes received (KB / s)
	NET_BYTE_OUT	Bytes sent (KB / s)
	Context_Switch	No. of context switches / s
Middleware Layer	BLOCK_RECEIVED	No. of blocks received / s
	TASK_LAUNCH	No. of tasks launched / s
	TASK_FINISH	No. of tasks finished / s
	TASK_DURATION	Average task duration (s)
	Comp. Time	Computation time
	Comm. Time	Communication time

- *Memory leaking:* On randomly selected nodes, the injected threads generate memory leaking on the nodes.
- *Frequent I/O operations:* On randomly selected nodes, the injected threads keep reading and writing a large amount of bytes from local files.
- *Network volume overflow:* On randomly selected nodes, the injected threads send and receive a large amount of data between nodes.
- *Deadlock:* On randomly selected nodes, the injected threads block MPI application processes from system resources.
- *Task Hang:* On randomly selected nodes, the injected threads deliberately miscompute checksum to trigger a hang at the reducer of Hadoop application.

We select these faults based on the literature and our own experience on system log analysis [45]. For example, we have found that some job delays are triggered by deadlock, some network-related problems like packet loss are caused by heavy traffic volume, and some performance problems are due to frequent I/O operations. While we generate all these faults from the operating system layer, they are used to simulate the manifestation of performance anomalies originated from different system layers.

5.3 Features

We collect 18 features per node. At the operating system layer, we collect node-level metrics from CPU, memory, I/O, and network by using four system commands, namely *vmstat*, *mpstat*, *iostat*, and *netstat*. At the middleware layer, we collect application-level metrics from the MPI workload by adopting profiling interface (PMPI) [46] and from the Hadoop workload by tracking the log file of *DataNode*. These features are summarized in Table 3. In experiments, the feature collection interval is set to 10 seconds and the detection interval is set to 30 seconds. Hence, in each detection, we collect three samples from every node and the matrix X described in Section 3.2 has a size of $(18 * 3) * n$.

5.4 Evaluation Metrics

We choose a widely used metric, *F-measure*, to evaluate detection accuracy. It is defined as

$$F - measure = \frac{2T_P}{2T_P + F_P + F_N}, \tag{8}$$

where T_P is the number of correctly detected abnormal nodes, T_N is the number of correctly detected normal nodes, F_P is the number of normal nodes detected as abnormal and F_N is the number of abnormal nodes detected as normal. It is worth noting that *F-measure* is a combined metric. Detection accuracy can be further evaluated by $sensitivity = T_P/(T_P + F_N)$ and $specificity = T_N/(F_P + T_N)$ respectively. A *sensitivity* of 1.0 means that the method identifies all the abnormal nodes, and a *specificity* of 1.0 means that the method spots all the normal nodes.

We also use *Rand Index* [47] to measure the accuracy of clustering. Specifically, given a group of nodes $A = (a_1, a_2, \dots, a_n)$, let P be the partition according to the clustering and G be the partition according to the true class labels, clustering accuracy (R) is defined as:

$$R = \frac{a + b}{a + b + c + d}, \tag{9}$$

where a is the number of pairs of elements in A that are in the same set in P and in the same set in G, b is the number of pairs in different sets in P and in different sets in G, c is the number of pairs in the same set in P and in different sets in G and d is the number of pairs in different sets in P and in the same set in G. *Rand index* is in the range of $[0, 1]$, with 0 indicating P and G do not agree on any pair of nodes and 1 indicating P and G are exactly the same. A good clustering should provide a high value (close to 1.0) for R.

6 Results

6.1 Detection Accuracy

In this set of experiments, we evaluate detection accuracy of our method under different workloads. The group size is set to 8, and the anomaly probability is set to 15 percent. Tables 4 and 5 present the results with single anomaly type and the combination of multiple anomaly types respectively. For single anomaly, the results (sensitivity & specificity) are the average detection accuracy of multiple experiments (i.e., multiple runs). For multiple anomalies, we randomly injected two or three or four anomalies out of the six anomaly types listed in Section 5.2 into the computing.

We make several observations from Tables 4 and 5. First, detection accuracy varies according to different anomaly types. Our method can achieve 100 percent sensitivity and high specificity for CPU hog, memory leaking and disk hog. This observation makes sense as we find that the features collected from CPU, memory and disk exhibit good similarity among healthy nodes. We also find that the false positive (i.e., specificity $<$ 1.0) is mainly caused by the variability of network features on normal nodes. With respect to network anomaly, detection accuracy is obviously lower than that with other anomaly types. The main reason is that the peer-comparable property of the network features collected in our experiments is interfered by factors such as application

TABLE 4
Detection Accuracy Measured by *Sensitivity* (Upper) and *Specificity* (Lower) with Single Anomaly Type

Single Anomaly	Stampede							HEC		
	CG	NB	MG	FT	MC	FL	NK	TS	BC	HJ
CPU	1.000.98	1.001.00	1.001.00	1.000.97	1.001.00	1.000.99	1.000.97	1.001.00	1.000.95	1.000.93
memory	1.000.99	1.000.99	1.000.98	1.000.98	1.001.00	1.001.00	1.000.98	1.001.00	1.000.98	1.000.96
I/O	1.000.99	1.001.00	1.001.00	1.000.99	1.001.00	1.000.00	1.000.99	1.001.00	1.000.98	1.000.98
network	1.000.97	1.000.94	0.980.92	0.960.88	1.000.99	1.000.97	1.000.96	0.980.95	0.920.83	0.890.76
deadlock	1.000.98	1.000.96	1.000.96	0.980.92	1.000.98	0.000.98	1.000.97	$n/an/a$	$n/an/a$	$n/an/a$
hang	$n/an/a$	$n/an/a$	$n/an/a$	$n/an/a$	$n/an/a$	$n/an/a$	$n/an/a$	0.980.95	0.940.90	0.920.87

Anomaly probability is set to 15 percent. Group size is set to 8. The anomaly type "deadlock" is only applicable to MPI workloads, and the anomaly type "hang" is only applicable to Hadoop workloads.

TABLE 5
Detection Accuracy Measured by *Sensitivity* (Upper) and *Specificity* (Lower) with Multiple Anomaly Types

Multiple Anomaly	Stampede							HEC		
	CG	NB	MG	FT	MC	FL	NK	TS	BC	HJ
two-anomaly	1.000.99	1.000.98	1.000.95	0.980.95	1.000.99	1.000.99	1.000.98	0.980.95	0.930.87	0.910.82
three-anomaly	0.990.96	1.000.94	0.980.92	0.950.91	1.000.98	1.000.97	0.980.95	0.930.90	0.880.81	0.860.77
four-anomaly	0.980.94	1.000.92	0.970.91	0.920.87	1.000.97	1.000.96	0.960.93	0.910.89	0.830.75	0.810.72

Anomaly probability is set to 15 percent. Group size is set to 8.

communication patterns, network environments, etc. For instance, in the MPI environment, FT has the lowest detection accuracy because it is a highly communication-intensive application, whose collectives/point-to-point ratio and communication/computation ratios are much higher than other MPI workloads. On the contrary, MC has the highest detection accuracy because it involves very few communications among nodes. With respect to deadlock and task hang, detection accuracy is higher than that with net hog but lower than that with other anomaly types. This observation is understandable as we note that the features collected from middleware layer show lower variability on normal nodes than the features collected from network.

We make similar observations from the two proxy applications. The detection method has slightly better performance on FL than on NK. This is mainly because the peer variability of network features from NK is significantly higher than that from FL. Specifically, NK features scalable all-to-all communication, which performs well on large-scale computing. On the contrary, FL is less communication intensive and its inter-process interactions are mainly through local point-to-point communications.

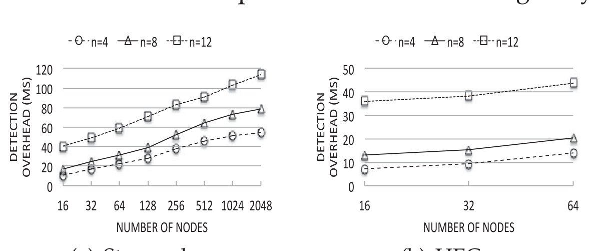
Second, we find the number of anomaly types has contradictory impacts on detection accuracy. Two-anomaly leads to a comparable or even better detection accuracy than one-anomaly. This observation accords with our analysis of the *best-case* and *worst-case* in Fig. 4. That is, multiple anomaly types are more distinguishable than single anomaly type using the majority voting mechanism. However, as the number of anomaly types increases to three and four, detection accuracy begins to drop because more noise are introduced.

Third, our method shows better performance on Stampede than on HEC. For the tightly-coupled scientific applications on Stampede, the number of anomaly types does not impact on detection accuracy significantly; while for the data-intensive applications on HEC, a larger number of anomaly types always indicates a lower detection accuracy. There are several explanations. On one hand, the tightly-coupled computing environment introduces less noise to node behaviors than the loosely-coupled environment. On the other hand, the MPI paradigm tends to impose more strict parallelism on compute nodes than the Hadoop paradigm, thus exhibiting a better peer-comparable property. In addition, as the HEC cluster used for our experiments is shared by other users, the normal node behaviors are interfered by resource contention.

6.2 Communication and Computation Cost

In this set of experiments, we study the scalability of our method by examining the overhead introduced by one detection. The sum of computation and communication time is measured in milliseconds (ms) with varying group sizes and number of nodes. Figs. 7a and 7b present the average detection overhead of different workloads on Stampede and HEC respectively. Each plot contains three curves, representing different group sizes.

We make two observations from Figs. 7a and 7b. First, given the same number of nodes, group size n impacts on detection overhead significantly. The main reason for this observation is that computation cost increases greatly as



(a) Stampede (b) HEC

Fig. 7. Average detection overhead with varying group size and number of nodes.

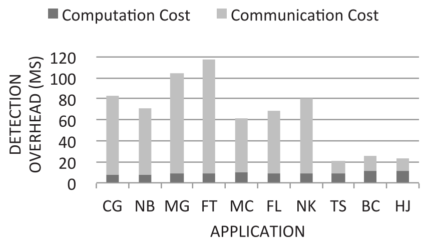


Fig. 8. Detection overhead under different workloads, where group size is set to 8. the number of nodes used on stampede is $2{,}048$ and that used on HEC is 64.

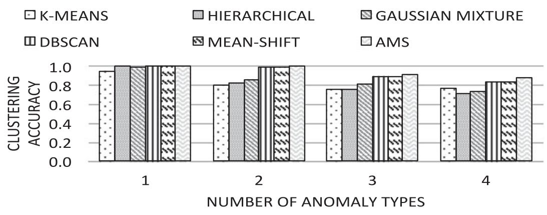


Fig. 9. Comparison of our non-parametric clustering method AMS and existing parametric clustering methods.

group size grows. This also implies that it is necessary to limit the group size if we want to apply complicated machine learning algorithms to data analysis within each group. Second, a larger number of nodes always results in a higher detection overhead. This is because communication cost grows significantly with the increase of number of nodes.

Fig. 8 further presents detection overhead under different workloads. On one hand, as we use the same group size for the test, the computation cost under different workloads are quite close. On the other hand, the communication cost under different workloads are impacted by the their communication patterns greatly. For instance, the FT algorithm which has heavy collective communications can lead to detection overhead two times higher than the MC algorithm which has very few communications. In sum, our anomaly detection method introduces a very low overhead into computing, indicating a high scalability. With the default group size 8, the overall time cost of each detection on $2{,}048$ compute nodes is less than 120 ms. Comparing with the detection interval set in our study (i.e., 30 seconds), the relative overhead is less than 0.4 percent.

6.3 Non-Parametric versus Parametric Clustering

In this set of experiments, we compare AMS with five well-known parametric clustering algorithms. They are k-means clustering, hierarchical clustering, Gaussian mixture clustering, DBSCAN and mean-shift clustering.

As mentioned in Section 3.3, parametric clustering methods require predefined parameters as inputs. In this study, we set the number of clusters to 2 for k-means, hierarchical and Gaussian mixture clustering by assuming there are two behavior patterns in the system. Also, based on the average peer distance and group size, we set the distance threshold to 0.2 and the minimum cluster size to 1 for DBSCAN, and set the bandwidth to 0.6 for mean-shift clustering. Figs. 9 and 10 present the comparison in term of clustering accuracy and detection accuracy respectively.

We make two observations from Fig. 9, where clustering accuracy is given by the average value under different anomaly probabilities. First, when the true number of behavior patterns is equal to the predefined number of clusters, i.e., number of anomaly types = 1, all clustering methods are comparable to each other; while as the number of anomaly types increases, the density-based clustering methods, including DBSCAN, mean-shift and AMS, outperform other clustering methods obviously because they are capable of identifying a arbitrary number of anomaly types. Second, AMS provides comparable detection accuracy as DBSCAN and mean-shift when the number of anomaly types is small; but slightly outperforms them when the number becomes larger. This observation indicates that AMS is more robust to the changing of data characteristics (e.g., the number of anomaly types) than those two widely adopted density-based clustering methods.

The comparison of detection accuracy shown in Fig. 10 basically coincides with the comparison of clustering accuracy shown in Fig. 9. Further, we find that k-means, hierarchical and Gaussian mixture clustering methods provide remarkably low detection accuracy for low anomaly probabilities (e.g., $p = 10\%$). The reason is that a low anomaly probability tends to form a large number of anomaly-free groups based on the random grouping strategy. As these three clustering algorithms divide nodes arbitrarily into two clusters, they result in a number of incorrect clusterings in these anomaly-free groups.

6.4 Effect of Feature Extraction Methods

In this set of experiments, we evaluate the effect of feature extraction methods on detection accuracy. Specifically, we compare three cases, including no feature extraction adopted, feature extraction by PCA and feature extraction by KCA. Fig. 11 presents the results.

As shown in Fig. 11, feature extraction does not impact on detection accuracy significantly when the number of anomaly types is less than 3. We guess this is because when the number of anomaly types is small, noise involved in data is limited. Thus the adoption of feature extraction dose

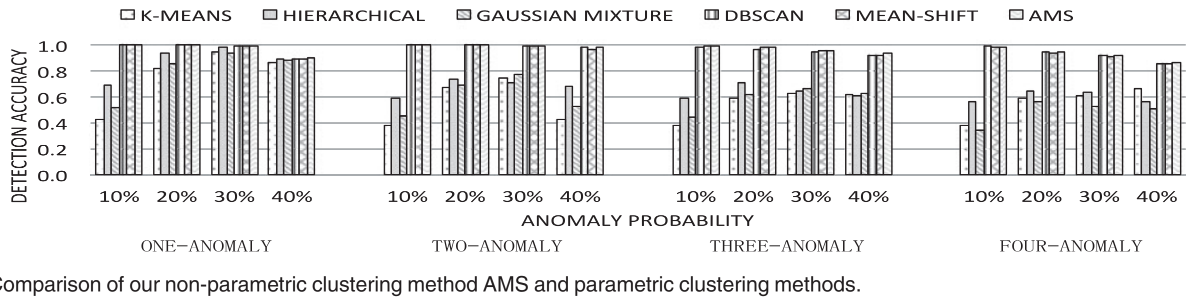


Fig. 10. Comparison of our non-parametric clustering method AMS and parametric clustering methods.

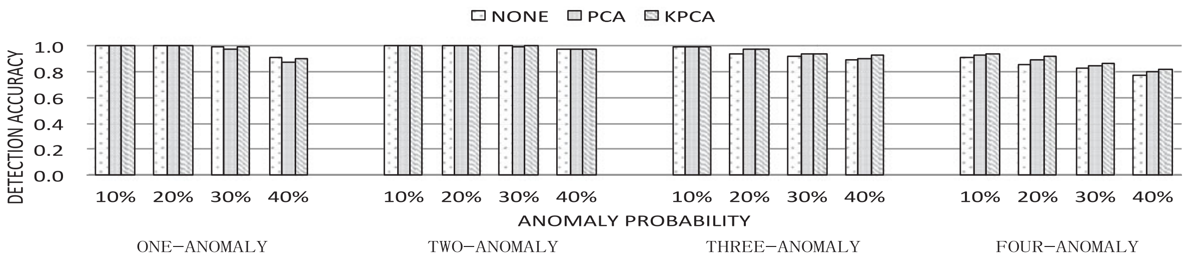


Fig. 11. Comparison of feature extraction methods in terms of detection accuracy.

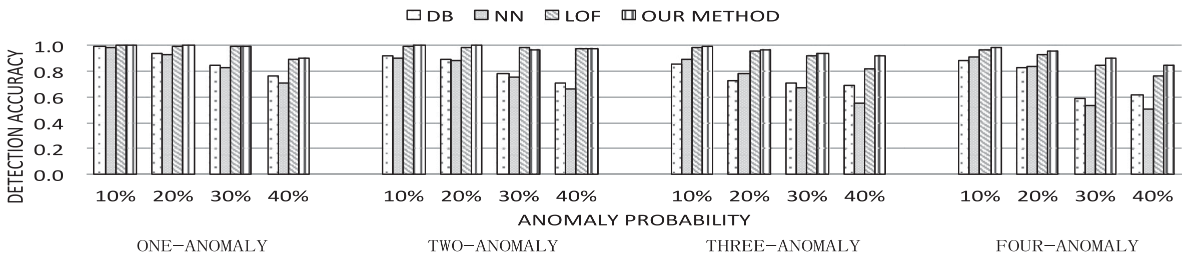


Fig. 12. Comparison of our detection method with existing detection methods on Stampede.

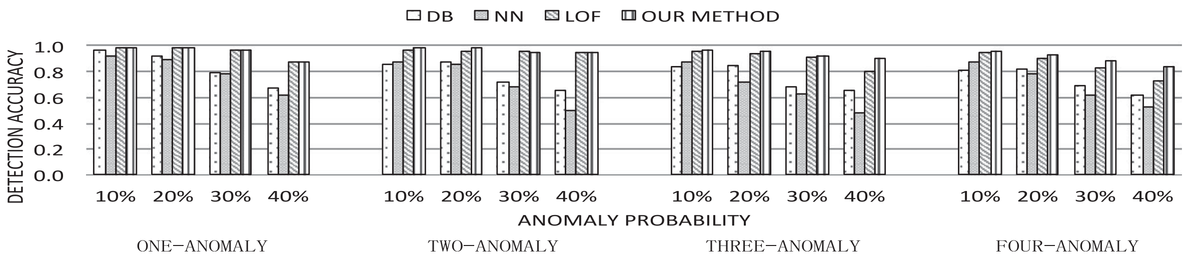


Fig. 13. Comparison of our detection method with existing detection methods on HEC.

not remove noise effectively but instead of miss some information. On the contrary, when the number of anomaly types is larger than 2, feature extraction improves detection accuracy effectively, and KPCA always provides higher detection accuracy than PCA. This indicates that KPCA can provide a better recognition rate than PCA in the context of complex behavior patterns. In addition, we find that if we limit group size not to be greater than the number of features per node, KPCA is computationally comparable to PCA.

6.5 Our Method versus Existing Methods

In this set of experiments, we compare our method with three widely used detection approaches. They are distance-based (DB) approach [14], nearest neighbor approach [48] and local outlier factor (LOF) based approach [49]. To make a fair comparison, we tune the parameters used in these methods carefully to achieve their best performance. Figs. 12 and 13 present the results on two testbeds respectively.

We make several observations from above figures. For a single anomaly type, detection accuracy of DB and NN are comparable to that of our method when anomaly probability is low, e.g., $p \leq 20\%$, but drops quickly as anomaly probability increases. For multiple anomaly types, DB and NN fail to achieve detection accuracy as high as our method even when anomaly probability is low. Unlike DB and NN, LOF is comparable to our method when the number of anomaly types is not larger than 2, but is outperformed by our method when it is larger than 2. The performance difference between these methods is mainly caused by their underlying models. As our method and LOF both rely on data density for anomaly detection, they have closer performance when compared to other two distance-based methods.

Compared to all other methods, our method has an average of more than 12 percent improvement in terms of detection accuracy, given multiple anomaly types (i.e., other than one-anomaly) and high anomaly probability (i.e., $p > 20\%$). We believe the improvement mainly comes from two aspects. First, the non-parametric clustering method distinguishes multiple anomaly types more accurately than existing methods that rely on predefined distance or density thresholds. Second, our two-phase voting strategy effectively boots detection accuracy on data with high anomaly probability.

7 Discussion

In this section, we discuss the limitations and the potential use of our design.

What happens after anomaly detection? Anomaly detection plays the first and a crucial step in fault management. The effectiveness of fault tolerant actions highly depend on detection accuracy. A false positive causes unnecessary error handling, while a false negative overlooks a potential performance anomaly that could lead to a system crash. In the past, numerous studies have been presented for error handling based on anomaly detection, and these include preemptive process migration, fault-aware job rescheduling, etc. [19], [50].

How should we select peer-comparable features? First of all, we shall point out that in this work we only require peer comparability per group, rather than across the entire system or the entire computing scale. While a production workload may not have such a peer-comparability property, the use of hierarchical grouping is intended to split the computing nodes into a number of smaller subgroups with a group size of 4, 8, or 12. By limiting the size of a group, it is more likely for preserving a peer-comparability property. Moreover, in the cases of dynamic workload behaviors, a performance monitor will be needed to dynamically determine when a new hierarchical grouping should be performed to reshape the node division for preserving the peer-comparability property.

Peer-comparable environment is one of the assumptions used in this study. In order to apply our method, appropriate features should be selected to characterize node behaviors to effectively represent the peer-comparable property. However, how to select appropriate features for anomaly detection is still a challenging problem. For instance, according to our experimental results, the features collected from CPU, memory and disk exhibit good peer-comparable property among normal nodes, while the features from network and application-layer are interfered by the workload dependent variability.

There are two ways to address the above issues. First, we can select more sophisticated features that are less vulnerable to the noise introduced by peer variability. For example, instead of using feature extraction to project the features into another space, we can deal with each feature separately, thus avoiding the situation where features with good peer-comparable property are interfered by features with poor peer-comparable property. Second, we can change our method from application-transparent to application-aware. For example, instead of using a fixed detection interval, the detection process can be divided into different stages according to the application's computation or communication pattern. By doing this, in each stage the nodes are expected to exhibit better peer-comparable property. However, adopting the second way will decrease the flexibility of our method and also require a deeper understanding of applications.

Can we apply our method to heterogeneous environments? The experiments in this work are conducted on two homogeneous clusters. The reason is that we do not have an access to a heterogenous cluster with a large number of nodes (e.g., hundreds to thousands of nodes). We are confident that our method works well in a heterogeneous environment by dividing heterogeneous nodes into separate groups through hierarchical grouping.

8 Related Work

Anomaly detection techniques can be broadly classified into two groups: model-based and data-driven. Representative model-based techniques include rule based methods [51], support vector machine (SVM) based methods [52], Bayesian network based methods [53], etc.

A number of model-based methods have been adopted for anomaly detection. For example, Tan et al. proposed a performance anomaly prevention method for virtualized cloud systems by building Markov chain models [2]; Wu et al. detected faults in cluster systems by comparing the current running state with normal running model [4]; Stewart et al. predicted application performance vis building system profile [7]; Bronevetsky et al. combined classification algorithms with information on the abnormality of application behavior to improve detection accuracy [54].

Our method is a data-driven approach, which makes detection using only the current system data. Data-driven methods that rely on node comparison for anomaly detection can be further categorized into nearest neighbor based and clustering based [21]. Representative nearest neighbor based methods include distance-based [14], kth nearest neighbor [48] and local outlier factor [49]. Representative clustering based methods usually use the well-known k-means algorithm [15].

A number of node comparison methods have been adopted for anomaly detection in large-scale systems. For example, Pertet et al. adopted peer comparison approach to identify anomalous nodes in actively replicated systems [10]; Kasick et al. developed anomaly detection mechanisms in distributed environments by comparing system metrics among nodes [9]; Ozonat adopted an information-theoretic approach to detect performance anomalies in distributed web services based on clustering [55].

Our study is distinguished from the above methods in two aspects. First, our detection method uses clustering analysis rather than deviation (threshold-based) to measure the peer similarity. Also, our method works in a decentralized manner and provides better scalability. Second, we assume the number of behavior patterns is unknown and adopt a non-parametric clustering method. It is more practical for realistic use.

Majority voting methods and their probabilistic models are widely studied for anomaly diagnosis in multiprocessor systems. Lee and Shin surveyed methods for automated probabilistic diagnosis of large multiprocessor systems [56]; Mourad and Nayak introduced a diagnosis approach using neural networks for fault identification using partial syndromes [57]; Mi et al. developed CloudDiag to efficiently pinpoint fine-grained causes of the performance problems without any domain-specific knowledge [58].

Distinguishing from above studies that use one-dimensional data and evaluate by simulation, this paper tackles high dimensional data from large-scale production systems. In addition, our design utilizes a new two-phase majority voting to boot detection accuracy in case of multiple anomaly types and high anomaly probability.

9 Conclusion

In this paper, we have presented a practical and scalable anomaly detection method for large-scale systems. Our design features a decentralized approach based on hierarchical grouping, non-parametric clustering, and two-phase majority voting. Experimental results have demonstrated that the proposed method can provide high detection accuracy by effectively distinguishing distinct anomaly patterns, with a negligible overhead. The proposed design is applicable to a variety of parallel and distributed computing environments with the peer-comparable property.

Our ongoing work includes evaluating the proposed detection method in other large scale computing environments. Further, we plan to study the issue of selection of features for anomaly detection. We will examine a variety of features obtained from different system layers and analyze their impact on detection under various parallel computing environments.

ACKNOWLEDGMENTS

This work was supported in part by US National Science Foundation grants CCF-1422009. Some preliminary results were presented in [59].

REFERENCES

[1] J. Tan, X. Pan, E. Marinelli, S. Kavulya, R. Gandhi, and P. Narasimhan, "Kahuna: Problem diagnosis for Mapreduce-based cloud computing environments," in *Proc. IEEE Netw. Operations Manage. Symp.*, 2010, pp. 112–119.

[2] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "PREPARE: Predictive performance anomaly prevention for virtualized cloud systems," in *Proc. IEEE 32nd Int. Conf. Distrib. Comput. Syst.*, 2012, pp. 285–294.

[3] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y. Wang, "Automatic misconfiguration troubleshooting with peerpressure," in *Proc. 6th Conf. Symp. Opearting Syst. Des. Implementation*, 2004, p. 17.

[4] L. Wu, D. Meng, W. Gao, and J. Zhan, "A proactive fault-detection mechanism in large-scale cluster systems," in *Proc. 20th Int. Conf. Parallel Distrib. Process.*, 2006, p. 97.

[5] K. Shen, C. Stewart, C. Li, and X. Li, "Reference-driven performance anomaly identification," in *Proc. 11th Int. Joint Conf. Meas. Model. Comput. Syst.*, 2007, pp. 85–96.

[6] H. Kang, H. Chen, and G. Jiang, "PeerWatch: A fault detection and diagnosis tool for virtualized consolidation systems," in *Proc. 7th Int. Conf. Autonomic Comput.*, 2010, pp. 119–128.

[7] C. Stewart and K. Shen, "Performance modeling and system management for Multi-component online services," in *Proc. 2nd Conf. Symp. Netw. Syst. Des. Implementation*, 2005, pp. 71–84.

[8] Z. Lan, Z. Zheng, and Y. Li, "Toward automated anomaly identification in large-scale systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 2, pp. 174–187, Feb. 2010.

[9] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan, "Black-box problem diagnosis in parallel file systems," in *Proc. 8th USENIX Conf. File Storage Technol.*, 2010, p. 4.

[10] S. Pertet, R. Gandhi, and P. Narasimhan, "Fingerpointing correlated failures in replicated systems," in *Proc. 2nd USENIX Workshop Tackling Comput. Syst. Problems with Mach. Learn. Techn.*, 2007, pp. 9:1–9:6.

[11] E. Perelman, M. Polito, J. Bouguet, J. Sampson, B. Calder, and C. Dulong, "Detecting phases in parallel applications on shared memory architectures," in *Proc. 20th Int. Conf. Parallel Distrib. Process.*, 2006, p. 88.

[12] M. Jiang, M. A. Munawar, T. Reidemeister, and P. Ward, "Automatic fault detection and diagnosis in complex software systems by Information-theoretic monitoring," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2009, pp. 285–294.

[13] I. Laguna, T. Gamblin, B. R. Supinski, S. Bagchi, G. Bronevetsky, D. H. Anh, M. Schulz, and B. Rountree, "Large scale debugging of parallel tasks with automaded," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 50:1–50:10.

[14] E. M. Knorr and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *Proc. 24rd Int. Conf. Very Large Data Bases*, 1998, pp. 392–403.

[15] D. Yu, G. Sheikholeslami, and A. Zhang, "Findout: Finding outliers in very large datasets," *Knowl. Inf. Syst.*, vol. 4, pp. 387–412, 2002.

[16] (2011). The tacc stampede website. [Online]. Available: https://www.tacc.utexas.edu/resources/hpc/stampede-technical

[17] J. Dean. (2008). Underneath the covers at Google: Current systems and future directions. [Online]. Available: http://atlantajobsin.blogspot.com/2012/06/google-io-2008-underneath-covers-at.html

[18] Apache Software Foundation. (2006). Apache's JIRA issue tracker. [Online]. Available: https://issues.apache.org/jira

[19] J. Smith, "A survey of process migration mechanisms," *ACM SIGOPS Operating Syst. Rev.*, vol. 22, no. 2, pp. 28–40, 1998.

[20] A. Avizienis, J. Laprie, and B. Randell, "Fundamental concepts of dependability," in *Proc. Inf. Survivability Workshop*, 2000.

[21] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, pp. 15:1–15:58, 2009.

[22] A. Quiroz, N. Gnanasambandam, M. Parashar, and N. Sharma, "Robust clustering analysis for the management of Self-monitoring distributed systems," *Cluster Comput.*, vol. 12, pp. 73–85, 2009.

[23] A. Quiroz, M. Parashar, N. Gnanasambandam, and N. Sharma, "Design and evaluation of decentralized online clustering," *ACM Trans. Auton. Adapt. Syst.*, vol. 34, pp. 1–31, 2012.

[24] I. T. Jolliffe, *Principal Component Analysis*. New York, NY, USA: Springer-Verlag, 1986.

[25] F. R. Bach and M. I. Jordan, "Kernel independent component analysis," *J. Mach. Learn. Res.*, vol. 3, pp. 1–48, 2003.

[26] S. Makeig, A. J. Bell, T.-P. Jung, and T. J. Sejnowski, "Independent component analysis of electroencephalographic data," in *Adv. Neural Inf. Process. Syst.*, 1996, pp. 145–151.

[27] F. Zhang, S. Lasluisa, T. Jin, I. Rodero, H. Bui, and M. Parashar, "In-situ feature-based objects tracking for large-scale scientific simulations," in *Proc. SC Companion*, 2012, pp. 736–740.

[28] A. Caruso, S. Chessa, P. Maestrini, and P. Santi, "Evaluation of a diagnosis algorithm for regular structures," *IEEE Trans. Comput.*, vol. 51, no. 7, pp. 850–865, Jul. 2002.

[29] B. Schölkopf, A. Smola, and K. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Comput.*, vol. 10, pp. 1299–1319, 1998.

[30] D. Comaniciu and P. Meer, "Mean shift analysis and applications," in *Proc. Int. Conf. Comput. Vis.*, 1999, p. 1197.

[31] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*. Wiley, 2011.

[32] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Rev.: Data Mining Knowl. Discovery*, pp. 231–C240, 2011

[33] B. Georgescu, I. Shimshoni, and P. Meer, "Mean shift based clustering in high dimensions: A texture classification example," in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, 2003, p. 456.

[34] D. Blough, S. Sullivan, and G. Masson, "Fault diagnosis for sparsely interconnected multiporcessor systems," in *Proc. Digest Papers., 19th Int. Symp. Fault-Tolerant Comput.*, 1989, pp. 62–69.

[35] M. Ding, D. Chen, K. Xing, and X. Cheng, "Localized Fault-tolerant event boundary detection in sensor networks," in *Proc. IEEE 24th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 2005, pp. 902–913.

[36] X. Luo, M. Dong, and Y. Huang, "On distributed Fault-tolerant detection in wireless sensor networks," *IEEE Trans. Comput.*, vol. 55, no. 1, pp. 58–70, Jan. 2006.

[37] (2014). The Top 500 List. [Online]. Available: http://www.top500.org/lists

[38] K. W. Schulz. (2013). Experiences from the deployment of TACC's stampede system. [Online]. Available: http://www.hpcadvisorycouncil.com/events/2013/Switzerland-Workshop/Presentations/Day_1/7_TACC.pdf

[39] (2012). NPB Website. [Online]. Available: https://www.nas.nasa.gov/publications/npb.html

[40] (2010). Barnes-hut Implementation on GitHub. [Online]. Available: https://github.com/robmdunn/nbody

[41] (2013). The monte carlo macroscopic cross section lookup benchmark. [Online]. Available: https://github.com/jtramm/XSBench

[42] (2014). FLASH. [Online]. Available: http://flash.uchicago.edu/site/flashcode

[43] (2014). NEK5000. [Online]. Available: https://github.com/Nek5000/Nek5000

[44] (2014). The apache hadoop website. [Online]. Available: http://hadoop.apache.org

[45] Z. Lan, J. Gu, Z. Zheng, R. Thakur, and S. Coghlan, "A study of dynamic Meta-learning for failure prediction in Large-scale systems," *J Parallel Distrib. Comput.*, vol. 70, pp. 630–643, 2010.

[46] V. Tabatabaee and J. Hollingsworth, "Automatic software interference detection in parallel applications," in *Proc. SC Conf.*, 2007, pp. 14:1–14:12.

[47] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *J. Am. Statist. Assoc.*, vol. 66, pp. 846–850, 1971.

[48] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 427–438.

[49] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 93–104.

[50] Y. Li, Z. Lan, P. Gujrati, and X.-H. Sun, "Fault-aware runtime strategies for high-performance computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 4, pp. 460–473, Apr. 2009.
[51] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian, "Anomaly extraction in backbone networks using association rules," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1788–1799, Dec. 2012.
[52] L. Khan, M. Awad, and B. Thuraisingham, "A new intrusion detection system using support vector machines and hierarchical clustering," *VLDB J.*, vol. 16, pp. 507–521, 2007.
[53] K. Das and J. Schneider, "Detecting anomalous records in categorical datasets," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2007, pp. 220–229.
[54] G. Bronevetsky, I. Laguna, B. R. de Supinski, and S. Bagchi, "Automatic fault characterization via Abnormality-enhanced classification," in *Proc. 42nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2012, pp. 1–12.
[55] K. Ozonat, "An Information-theoretic approach to detecting performance anomalies and changes for Large-scale distributed web services," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. with FTCS and DCC*, 2008, pp. 522–531.
[56] S. Lee and K. G. Shin, "Probabilistic diagnosis of multiprocessor systems," *ACM Comput. Surv.*, vol. 26, pp. 121–139, 1994.
[57] E. Mourad and A. Nayak, "Comparison-based system-level fault diagnosis: A neural network approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 6, pp. 1047–1059, Jun. 2012.
[58] H. Mi, H. Wang, Y. Zhou, M. R. Lyu, and H. Cai, "Toward Fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1245–1255, Jun. 2013.
[59] L. Yu and Z. Lan, "A scalable, Non-parametric anomaly detection framework for hadoop," in *Proc. ACM Cloud Autonomic Comput. Conf.*, 2013, pp. 22:1–22:2.

Li Yu received the BS degree from Sichuan University in 2004 and the MS degree from the Rochester Institute of Technology in 2009, respectively. He is currently working toward the PhD degree in computer science at the Illinois Institute of Technology since 2010. His research interests include HPC data analytics and performance modeling in large-scale systems. He is a student member of the IEEE.

Zhiling Lan received the PhD degree in computer engineering from Northwestern University in 2002. She has since joined the faculty of the Illinois Institute of Technology and is currently a professor at the Department of Computer Science. She is also a guest research faculty at the Argonne National Laboratory. Her research interests are in the areas of parallel and distributed systems, with particular emphasis on fault tolerance, power efficiency, resource management and job scheduling, performance analysis and optimization. She is a senior member of the IEEE and IEEE computer society.