Design and Evaluation of an Online Anomaly Detector for Distributed Storage Systems

Xin Chen[1], Xubin He[2], He Guo[3] and Yuxin Wang[4]
[1]Department of Electrical and Computer Engineering, Tennessee Technological University, Cookeville, TN, USA
[2]Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA, USA
[3]School of Software Technology, Dalian University of Technology, Dalian, LiaoNing, China
[4]School of Computer Science and Technology, Dalian University of Technology, Dalian, LiaoNing, China
Email: xchen21@students.tntech.edu, xhe2@vcu.edu, {guohe, wyx}@dlut.edu.cn

***Abstract*— Performance problems, which may stem from different system components, such as network, memory, and storage devices, are difficult to diagnose and isolate in distributed storage systems. In this paper, we present a performance anomaly detector which is able to efficiently detect performance anomaly and accurately identify the faulty sources in a system node of a distributed storage system. Our method exploits the stable relationship between workloads and system resource statistics to detect the performance anomaly and identify faulty sources which cause the performance anomaly in the system. Our experimental results demonstrate the efficiency and accuracy of the proposed performance anomaly detector.**

***Index Terms*— performance anomaly detector, distributed storage systems, parallel file systems**

I. Introduction

Performance is critical in the study of distributed storage systems. Synthetic workloads or file system benchmarks are created to examine the behaviors of storage systems. Although they are very useful in the initial stage of the design and development of storage systems, it is insufficient for using them to analyze or resolve one common problem called performance anomaly in these systems [1], [2]. By performance anomaly it means that the observed system behaviors are not expected according to the observed system workload. For example, I/O throughput has a significant degradation given a moderate amount of I/O requests. Performance anomaly is closely related to either some resource-intensive processes that demand large portion of system resources (CPU or memory) or some unexpected software and hardware behaviors like software bugs (memory leaking) and hardware faults (bad hard drive sectors), and it is common in storage systems. However, it remains a challenging task to efficiently detect performance anomaly and accurately identify the faulty sources, particularly in distributed storage systems.

Distributed storage systems usually consist of a large amount of commodity computer nodes which may have different processing capabilities. However, the overall performance of such systems is not determined by the fastest computer nodes of the systems, instead, the performance is often limited by the capability of the slowest ones [2], [3]. So, if there exists performance anomaly in some node of a distributed storage system, it is highly possible that the overall system performance will suffer negative effects, and such effects may be accumulated and magnified due to long-running and large-scale computations [2], which directly hurts the reliability and availability of the system. Therefore, it is necessary and crucial to equip distributed storage systems with a tool which is able to efficiently detect performance anomaly and accurately identify the faulty sources.

As compared to the fail-stop failures [4], it is more difficult to detect the existence of performance anomaly, and even more difficult to identify the source of the anomaly, because both dynamic workload change and many uncertain factors such as caching and scheduling can perplex people's ability to understand the system behaviors. Currently, some anomaly detecting approaches are threshold-based, which set thresholds for observed system metrics and raise signals when the thresholds are violated [5], [6]. However, it is difficult to choose appropriate thresholds for a variety of workloads and computer nodes with different capabilities. Some approaches are model-based, which indicate performance anomaly by comparing the observed system measurements and the model estimations [2], [7], [8], however, their usages are limited to the generality of the models.

This work targets the runtime diagnosis of performance anomaly in distributed storage systems which may consist of heterogeneous computer nodes and experience dynamic changed workloads. The proposed approach is self-diagnosis based, which exploits some invariants that exist in a computer node of a distributed storage system to detect the performance anomaly and identify faulty sources of that node. Such invariants refer to the stable relations between workloads and system resource statistics in faulty-free situations.

The rest of the paper is organized as follows: Section II gives a brief discussion on related work. In section III, we describe our methodology for performance anomaly detection and identification, and present the design of our performance anomaly detector in section IV. Section V describes our experiments and lists experimental results. Finally, we conclude the paper in section VI.

doi:10.4304/jsw.6.12.2379-2390

II. Related Work

For large-scale systems like cluster file systems, it is a major challenge to understand system behaviors, particularly unexpected behaviors. Numerous techniques have been proposed for detecting system anomalies. Among them, the simplest ones are the threshold-based techniques which are a form of service level agreements (SLAs). They are very useful on the condition that their users clearly know the key metric to monitor and the best value of the thresholds in different scenarios [5], [6]. Unfortunately, it is very difficult, even for an expert, to correctly choose the necessary metrics to monitor and set the right values of the thresholds for different scenarios in the context of today's complex and dynamic computer systems.

Recently, statistical learning or data mining techniques are widely employed to construct probability models for detecting various anomalies in large-scale systems based on some heuristics and assumptions, although these heuristics and assumptions may only hold in some particular systems or scenarios.

Kasick et al [2] developed a statistical peer-comparison diagnosis approach to identify a faulty node in a cluster file system. The rationale of their approach is based on the observation that there is an obvious difference between the behaviors of fault-free and faulty nodes. Kavulya et al [9] and Lan et al [8] proposed the similar approaches to detect performance problems in replicated file systems and a cluster system, respectively. However, the validation of these approaches is based on a strong assumption of homogeneous hardware and workloads, which may only hold in a few cases.

Besides the probability models for system metrics such as throughput, response time, etc, various relationships and correlations among system inputs and measurements are also explored and modeled to detect anomalies in large-scale computer systems. Chen et al [7] developed a new technique, the principal canonical correlation analysis (PCCA), to perform failure detection in large-scale computer systems which provide online Internet services. The key idea of their approach is to capture the contextual relationships between the system inputs and their internal measurements which hold in fault-free scenarios, and are broken in faulty scenarios. However, it is required for applying their technique that there exists a linear relationship between the system inputs and their internal measurements.

Guo et al [10] and Gao et al [11] investigated the probabilistic correlation between flow-intensities measured at different points and the one between different system measurements, respectively. In this work, we also exploit the correlation among system measurements, however, we not only use them to detect the existence of performance anomaly in a cluster file system, but also pinpoint the source of the performance anomaly.

III. Performance Anomaly Detection and Identification

Given the scale and heterogeneity of distributed storage systems, it is usually difficult to perform peer comparison to distinguish faulty nodes from fault-free nodes, because the behaviors of nodes in these systems may not be comparable. In this work, a self-diagnosis based approach is adopted to detect the existence of performance anomaly and identify the faulty resources. The major advantage of the approach is its independence of the scale and heterogeneity of distributed storage systems.

The feasibility of the approach is based on two observations. First, resource overuse (CPU and memory) and hard disk faults are very common in today's distributed storage systems according to the recent studies [12]–[19]. They manifest themselves at least on a computer node. Thus, if it is able to identify the system abnormal behaviors originated from them by analyzing system measurements collected on a computer node, it is not necessary to adopt centralized or peer comparison based performance anomaly detectors, which are expensive and not practical for heterogeneous computer systems.

Second, there exist some relations among the system measurements of a computer node in distributed storage systems, which can be regarded as invariants when the node works properly; but, one or more of such invariants does not hold once the system experiences performance anomaly [10], [20]. Such observation lays a strong foundation for performing self-diagnosis based performance anomaly detection and faulty resource identification, because performance anomaly can be detected and faulty sources can be identified by simply checking whether some invariants hold or not. Therefore, the main task is to figure out the invariants of distributed storage systems which can work as an indicator of performance anomaly.

A. Relation among Computer Nodes in Distributed Storage Systems

Before exploring the invariants of distributed storage systems, it is necessary to understand the relation among computer nodes in these systems. These distributed storage systems typically consist of three main components: clients, a metadata server or a server cluster (MDS), and a cluster of I/O servers or object storage devices (OSDs). They provide an inexpensive alternative utilizing Commodity Off The Shelf (COTS) products allowing large I/O intensive applications to be run on high performance clusters [21]. Figure 1 presents a general architecture of such systems. In this architecture, metadata operations are separated from I/O operations, and there exist two types of relations among system nodes: the relation between a metadata server and multiple I/O servers and the relation among a set of I/O servers.

The first type of relation reveals a single point of failure in these storage systems. Because metadata servers are always accessed before actual data transferring, once metadata servers are down, clients cannot initiate any I/O operations. On the other side, metadata severs do not

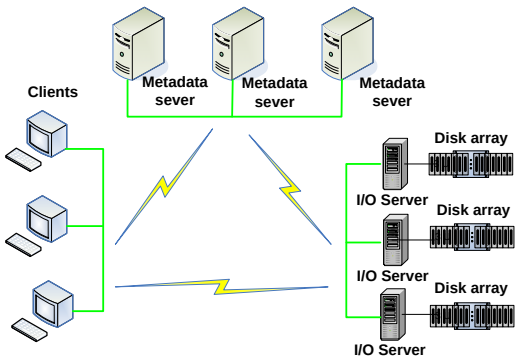


Figure 1: The general architecture of distributed storage systems.

intervene the actual I/O processing between clients and I/O servers. Even if a metadata sever is down during an I/O operation, the I/O operation can still be completed [22]. In this work, because the concentration is I/O performance of a distributed storage system, the relation between a metadata server and multiple I/O servers is not considered. The main focus is on studying the relation among a set of I/O servers which is more relevant to system I/O performance.

In a distributed storage system which deploys file systems like PVFS [23] and Lustre [24], data are usually distributed among a set of I/O servers. An I/O request from a client normally consists of several sub-I/O requests corresponding to different I/O servers. When a client processes such an I/O request, it is blocked on some syscall until it properly receives all I/O server responses, and then it can process next I/O request. Figure 2 presents an example of an I/O request sequence from a client. In this example, there are total three I/O requests, and each I/O request contains three sub-I/O requests. For example, sub-I/O request 1, 2, and 3 are of the first I/O request from the client, and they are issued to I/O server 1, 2, and 3, respectively. Let s_1, s_2, and s_3 denote the request sending rate of the client to I/O server 1, 2, and 3 respectively, and they are calculated by Formula 1.

$$\begin{cases} s_1 = \dfrac{\text{I/O request 1 + I/O request 4 + I/O request 7}}{t_3 - t_0} \\ s_2 = \dfrac{\text{I/O request 2 + I/O request 5 + I/O request 8}}{t_3 - t_0} \\ s_3 = \dfrac{\text{I/O request 3 + I/O request 6 + I/O request 9}}{t_3 - t_0} \end{cases} \quad (1)$$

Thus, when an I/O server experiences a performance problem, it definitely increases the client's response time

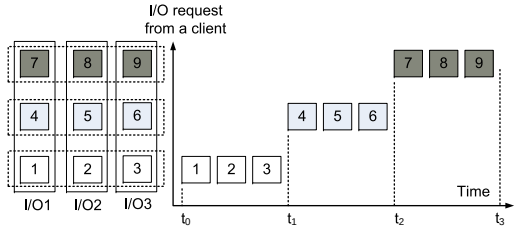


Figure 2: An I/O request sequence from a client.

(service time + waiting time), and as a result, s_1, s_2, and s_3 will decrease due to the increase of $t_3 - t_0$. Because the request receiving rate of an I/O server is proportional to the sending rate of a client, an important relation among a set of I/O servers can be concluded that once an I/O server experiences a performance problem (e.g., I/O performance decrease), it receives fewer requests per time unit from clients, so do other I/O servers.

Figure 3 shows I/O request receiving rates of a normal node and faulty node in the presence of a performance anomaly. It is clear that when a performance problem occurs at a computer node, the problem not only reduces the amount of received request per second at the faulty node, but also manifests similar symptom at other nodes which work properly.

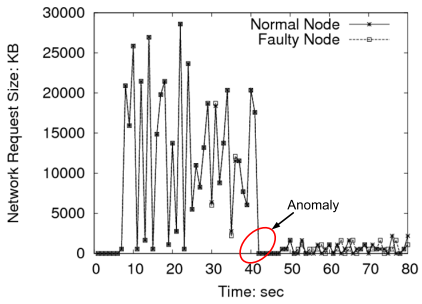


Figure 3: I/O request receiving rate during a sequential write. At the 41st second, disk delay faults were injected in a computer node, which produced a performance anomaly.

B. *Invariants*

Various relations among system measurements exist in a computer node of a distributed storage system. Here, invariants refer to those stable relations when a system properly works. Because I/O performance is very important in distributed storage systems, in this work, the focus is on how to discover and utilize invariants in a computer node to detect and pinpoint I/O performance problems.

Because any performance problem at a computer node manifest symptoms of unexpected certain resource usage, because system resources are always limited, once one or more processes occupies too many resources and does not release them, the executions of other processes are negatively impacted, as the OS kernel forces the processes sleep until the required resources are ready [25]. Meanwhile, if a resource request from a process cannot be satisfied immediately, the kernel also forces the process sleep. Thus, one option of utilizing invariants in a computer node to detect performance anomaly is to explore the relations between workloads and system resource statistics.

To facilitate the discussion, how a computer node handles the I/O requests from clients is first studied. Figure 4 depicts an I/O request flow in a computer node. External I/O requests are first processed by the process

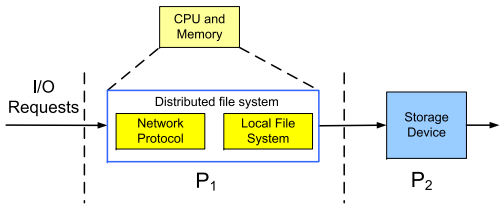


Figure 4: An I/O request flows in a computer node.

of a distributed file system on the node, then the process sends the I/O requests to a hard disk, and the hard disk finally completes those I/O requests. According to the location of I/O requests, the flow can be divided into two phases: P_1 and P_2. The former indicates the phase where I/O requests are processed by the distributed file system, and the latter represents the phase where I/O requests are in a hard disk.

In the two phases, different resources are required for processing incoming I/O requests. CPU and memory are two major required resources in P_1, as in that phase, a distributed file system transforms the I/O requests from network into the ones for the local disk. I/O requests are finally stratified in P_2, the local disk is the major required resource in that phase. Thus, the concentration is of analyzing the relations between workloads and the statistics of the resources listed above to look for the invariants which can be used to detect and pinpoint performance problems. By studying the trace data collected from the previous studies on distributed storage systems [3], [22], three invariants are concluded as follows based on the statistics listed in Table I.

Invariant for memory. If the process of a distributed file system at a computer node works properly, without intervention of other processes, the total size of I/O requests over network per second is proportional to the amount of the allocated memory per second.

Memory is allocated to hold data either after the arrival of write requests from clients or before sending back the satisfied read requests to clients. Thus, if a computer node has sufficient free memory and there are no other memory intensive processes running on the node, the total size of I/O requests over network per second is proportional to the amount of the allocated memory per second. The invariant is used to identify the performance problems originated from memory. Figure 5 gives an example of the invariant.

Invariant for CPU. If the process of a distributed file system at a computer node works properly, without intervention of other processes, the total size of I/O requests over network per second is proportional to the number of interrupts per second.

Interrupts are generated during the processing of I/O requests. For example, a network interface card raises hardware interrupt to CPU after the arrival of I/O requests from clients; disk interrupts are triggered when I/O requests are issued to a hard disk drive. If more I/O requests arrive at a computer node, more interrupts are generated, and vice versa. Meanwhile, the generation rate

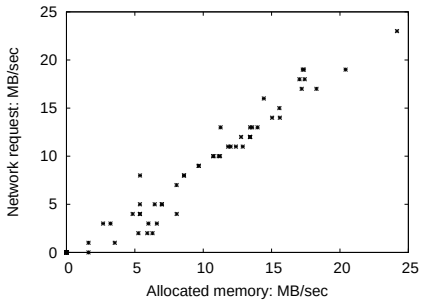


Figure 5: The relation between the total size of I/O requests over network per second and the amount of allocated memory per second. Data is from a trace of 40 seconds I/O activities in a computer node.

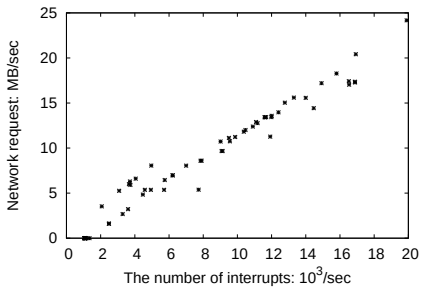


Figure 6: The relation between the total size of I/O requests over network per second and the number of interrupts per second. Data is from a trace of 40 seconds I/O activities in a computer node.

of interrupts is closely related to the CPU time of the corresponding process, as it requires a significant amount of CPU time to process I/O related interrupts [26]. Once the CPU resource is insufficient for the distributed file system process, fewer I/O related interrupts are generated, and the proportional relation between I/O request arrival rate and interrupt generating rate does not hold. The invariant is used to identify the performance problems originated from CPU. Figure 6 gives an example of the invariant.

Invariant for disks. If a hard disk works properly and has continued I/O requests, the average I/O request size is proportional to the average I/O request service time.

I/O requests issued to a hard disk usually have different sizes. It is intuitive that larger requests require more service time than smaller requests. However, when hard disks process discontinued I/O requests, small requests may require more service time than large requests, because the disk seek time dominates the total request service time. Thus, when a hard disk works properly and has continued I/O requests, the average I/O request size is proportional to the average I/O request service time. The invariant is used to identify the performance problems originated from hard disks. Figure 7 gives an example of the invariant. In the figure, the proportional relation is maintained among I/O requests with large size, but if I/O request size is very

TABLE I.: Workload and System Resource Statistics.

Source	Metric	Description
Workloads	net_req	the total size of I/O requests over network per second.
	avgrq-sz	The average size (in sectors) of the requests that are issued to a disk drive.
System resource	free_mem	The amount of idle memory.
	in	The number of interrupts per second, including the clock.
	svctm	The average service time (in milliseconds) for I/O requests that are issued to a disk drive.

small, the proportional relation rarely holds.

C. Indicators

According to the relation among I/O servers concluded in Section III-A, once an I/O server has a performance problem, the problem will be also observed at other I/O servers. Such a relation makes it difficult to accurately locate the faulty server. Furthermore, dynamically changing workloads perplex people's ability to determine appropriate thresholds to identify performance anomaly [1]. It is necessary to find indicators which are highly sensitive to performance anomalies but less sensitive to other factors. In this work, the invariants discussed in the previous section are leveraged to develop such indicators.

1) Indicators of Performance Anomalies: Although one or more of the above invariants does not hold when an I/O server experiences performance problem, it is still insufficient to only depend on them to detect the existence of performance anomaly on the server, because even if when an I/O server works properly, these invariants may still not hold, for example, marginal memory allocation by other processes may break the invariant for memory but does not negatively impact the running of the process of a distributed file system on the server.

To compensate the drawback of the invariants, an indicator $I_{req}(n)$ is adopted to detect the performance anomaly on an I/O server at the nth sampling period. Formula 2 gives the definition of $I_{req}(n)$, where req_{n-1} denotes the average total size of I/O requests at the $(n-1)$th sampling period, req_n denotes the average total size of I/O requests at the nth sampling period, and α denotes a threshold of the degradation ratio between req_{n-1} and req_n. If the ratio is greater than or equal to α, $I_{req}(n)$ generates a TRUE value, which suggests a performance problem, otherwise not. Similarly, $I_{req}(n)$ cannot be used alone to detect performance anomaly on an I/O server, because non-faulty I/O servers also observer the degradation of receiving request rate. $I_{req}(n)$ should be combined with the indicators of the invariants to detect performance anomaly.

$$I_{req}(n) = \begin{cases} \text{FALSE, if } \frac{req_{n-1}-req_n}{req_n} < \alpha, req_n \neq 0 \\ \text{FALSE, } req_n = 0 \\ \text{TRUE, if } \frac{req_{n-1}-req_n}{req_n} \geq \alpha, req_n \neq 0 \end{cases} \tag{2}$$

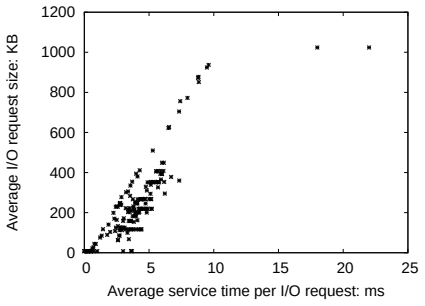


Figure 7: The relation between average I/O request size and average service time per I/O request. Data is from a trace of 250 seconds I/O activities on a computer node.

2) Indicators of Faulty Sources: Because the invariants discussed above refer to a proportional relation between two metrics, in order to use them in practice, such a proportional relation needs to be quantified. The correlation $corr(x, y)$ is a good measurement for quantifying a proportional relation between two variables: x and y. Formula 3 gives a formal definition of $corr(x, y)$, where $\sigma_{x,y}$ denotes the covariance of x and y; σ_x and σ_y denote the variance of x and y, respectively; μ_x and μ_y represent the mean value of x and y, respectively; $E(x)$ calculates the expectation of variable x. The sign of $corr(x, y)$ is more meaningful than its absolute value: once correlation is positive, it indicates x increases as the increase of y; otherwise, it indicates x is not proportional to y.

$$corr(x,y) = \frac{\sigma_{x,y}}{\sigma_x \sigma_y} = \frac{E[(x-\mu_x)(y-\mu_y)]}{\sqrt{E(x-\mu_x)^2}\sqrt{E(y-\mu_y)^2}} \tag{3}$$

Thus, based on Formula 3, three indicators I_{mem}, I_{cpu}, and I_{disk} are defined to test the invariants by Formula 4, 5, and 6, respectively. If an indicator has a boolean value of TRUE, the corresponding invariant holds, otherwise, the invariant does not hold, which suggests the performance problem originates from the corresponding resource. Table II lists the parameters used in these formulas.

TABLE II.: Symbols in Formulas 4, 5, and 6.

Parameter	Description
req	the total size of incoming I/O requests per second.
$interrupt$	the number of generated interrupts per second.
mem	the amount of allocated memory per second.
$iosize$	the average I/O request size to a hard disk per second.
$svctm$	the average I/O request service time per second.

TABLE III.: An Example of a Probability Distribution Table of $P(iosize|svctm)$. Data is from a trace of 250 seconds I/O activities in a computer node.

I/O request size: KB	Average service time: ms		
	[0, 8)	[8, 12)	[12,)
[0, 150)	100%	0	0
[150,)	85%	10%	4%

$$I_{cpu} = \begin{cases} \text{FALSE, if } corr(req, interrupt) < 0 \\ \text{TRUE, if } corr(req, interrupt) \geq 0 \end{cases} \quad (4)$$

$$I_{mem} = \begin{cases} \text{FALSE, if } corr(req, mem) < 0 \\ \text{TRUE, if } corr(req, mem) \geq 0 \end{cases} \quad (5)$$

$$I_{disk} = \begin{cases} \text{FALSE, if } corr(iosize, svctm) < 0 \\ \text{TRUE, if } corr(iosize, svctm) \geq 0 \end{cases} \quad (6)$$

Because I_{disk} does not work well in the case of discontinued I/O requests, in order to compensate the drawback of I_{disk}, the conditional probability distribution $P(iosize|svctm)$ is checked to confirm the value generated by I_{disk}; as if a hard disk works stable, a stable probability distribution has to be observed. Table III gives an example of the probability distribution. For example, if a sequence of {(80KB, 3ms), (81KB, 2ms), (82KB, 4ms), (83KB, 2ms)} is observed, although I_{disk} generates a FALSE value, the $P(iosize|svctm)$s of all pairs in the sequence are 100% according to Table III, thus, the FALSE value of I_{disk} is not confirmed and a TRUE value is generated. In this work, once I_{disk} generates a FALSE value, and the corresponding $P(iosize|svctm)$ is less than 10%, the FALSE value of I_{disk} is confirmed, otherwise, I_{disk} generates a TRUE value.

Although higher accuracy can be achieved by checking $P(iosize|svctm)$ for identifying disk problems than only looking at the proportional relation between $iosize$ and $svctm$, the major drawback of the method is the long training time which is required for collecting sufficient data to calculate a dependable probability distribution. So, I_{disk} serves as the major indicator of disk problems in the absence of a dependable $P(iosize|svctm)$.

As compared to the performance problems originated from memory, CPU, and hard disks, the problems from network are more difficult to diagnose, as they usually manifest themselves as a symptom of workload change, and it is difficult to only use the local information of an I/O server to identify them. An indicator $I_{network}$ is defined by Formula 7, which combines the local information of an I/O server and the information from other related I/O servers to identify the network problems. In Formula 7, $I^n_{network}$ is a local indicator of network on an I/O server n, its TRUE value suggests there may have some network problem which causes the performance anomaly, but the value should be confirmed by the external information from other I/O servers; $I'_{network}$ finally determines whether the network is a faulty source or not, if a TRUE value is generated by it, the source of performance anomaly can be pinpointed to the network.

$$\begin{cases} I^n_{network} = I^n_{disk} \wedge I^n_{mem} \wedge I^n_{cpu} \wedge I^n_{req},\ n \in N \\ I'_{network} = I^1_{network} \wedge I^2_{network} \wedge \cdots \wedge I^n_{network},\ n \in N \end{cases} \quad (7)$$

IV. The Design of the Online Performance Anomaly Detector

The online performance anomaly detector is implemented as a daemon process which runs at each computer node of a cluster file system. The detector sends alarms to clients or administration nodes, when performance anomaly is detected at a computer node. It is worth pointing out that once performance anomaly is detected on a computer node, it is most likely that the other computer nodes generate alarms soon, and those alarms may mark other resource as faulty, meanwhile, one or more of our invariants on the computer node may not hold any more until the performance anomaly is fixed. Thus, the alarms raised after the first alarm in a short period are ignored.

Figure 8 shows the working flow of our performance anomaly detector. The detection process is triggered when there is a significant degradation of req, then all indicators are evaluated accordingly to identify which system component is the faulty source, finally an alarm is raised if the performance anomaly is detected.

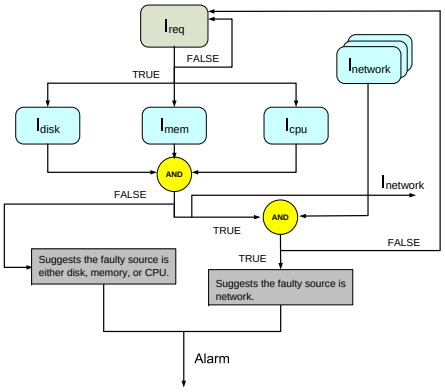


Figure 8: The working flow of the online performance anomaly detector.

V. Experiments

To demonstrate the efficiency of our performance detector, we constructed a testbed which consisted of four computer nodes (1 metadata server, 3 I/O servers). These servers have different computation and I/O capabilities, as shown in table IV. Our detector was evaluated with synthetic workloads on a parallel file system, PVFS. Four faults were injected to produce faulty situation during the evaluation: disk delay faults, network delay faults, CPU overuse faults, and memory overuse faults. disk delay faults introduce extra I/O request processing time in a hard disk driver; network delay faults add extra delay at an I/O server for sending every request over the network; CPU and memory overuse faults limit the available CPU

and memory resource at a low level, respectively. In our experiments, we adopted a sampling period of four seconds according to our prior experience, in which four samples were taken, one per second, and all indicators were evaluated at the end of the period; we set α to 50% for I_{req}.

In order to measure the efficiency and accuracy of our detector, two metrics are defined: the detection latency and the true positive rate. The former measures how long our detector may take to detect the existence of performance anomaly after the injection of performance faults, and the latter measures the accuracy of our detector in terms of the percentage of correct alarms. Formula 8 and 9 give the definitions of the two metrics, where Δ denotes the detection latency, T_d represents the time point at which performance anomaly is detected, T_i denotes the fault injection time point, A_{td} denotes the true positive rate, N_{td} and N_{fd} represent the number of true and false detections, respectively.

$$\Delta = T_d - T_i \tag{8}$$

$$A_{td} = \frac{N_{td}}{N_{td} + N_{fd}} \tag{9}$$

In this section, the behaviors of our performance anomaly detector are examined with synthetic workloads in different faulty situations. Before the discussion of our detector in faulty situations, the system behaviors in fault-free situation are studied first; we focus on examining whether our invariants hold or not, which is of ultra importance for the correctness of our detector.

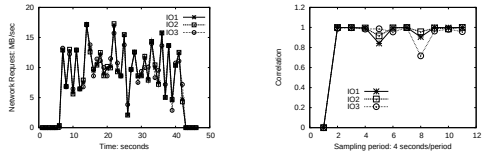
Figure 9 shows the results of 1GB sequential write tests on PVFS in fault-free situation. In the these figures, our three invariants perfectly hold in the presence of a significant fluctuations of external I/O request rate for both file systems, as the values of three correlations along the time axis are almost positive. The only exception is in figure 9d, the correlation of $iosize_n$ and $svctm_n$ is negative at the second sampling period. However, it is reasonable, as in the period, I/O servers just started to process I/O requests, hard disks may take relative long service time for processing the first incoming I/O requests with moderate sizes, which breaks the third invariant.

The results of 1GB sequential read tests on PVFS in fault-free situation are shown in figure 10. As similar as in figure 9, the invariants for memory and CPU hold through the tests, but the invariant for disk does not always hold, as there is no data caching for PVFS, which results in discontinuous I/O requests. Because there is no significant drop of req in figure 10, even if the invariant is broken, no alarm is raised by our detector in practice.

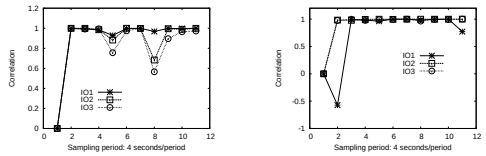
Due to the space limit, we only discussed the results of write tests of the following experiments, and gave a summary of both write and read tests in section V-F.

A. Disk delay faults

This set of experiments evaluated our performance anomaly detector in the case of disk delay faults which do not fail any I/O request but introduce extra I/O request

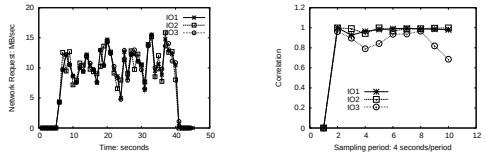


(a) External I/O request rate (b) $corr(req, mem)$

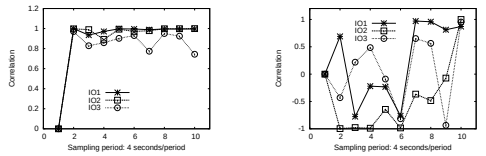


(c) $corr(req, interrupt)$ (d) $corr(iosize, svctm)$

Figure 9: 1GB sequential write on PVFS.



(a) External I/O request rate (b) $corr(req, mem)$



(c) $corr(req, interrupt)$ (d) $corr(iosize, svctm)$

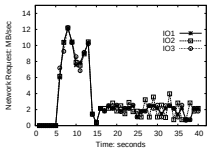
Figure 10: 1GB sequential read on PVFS.

processing time in a hard disk driver. The delay was set to 50 ms for the following experiments. Figure 11 shows the results of 1GB sequential write test on PVFS where the disk delay faults were introduced at the 4th sampling period (13rd – 16th second) at IO2.

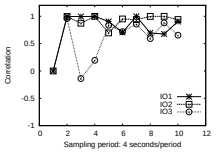
In figure 11, although the invariants for memory and CPU of IO3 do not hold at the 3rd sampling period, req of IO3 does not have a significant drop during such the period which is between the 9th and 12rd second in figure 11a, thus, there was no alarm raised. In the 13rd second, disk delay faults were introduced at IO2, we not only observed a sharp drop of req but also saw the FALSE value generated by I_{disk} of IO2 in the 4th sampling period which includes the 13rd second time point, meanwhile, at the same sampling period, no other invariant was broke. Thus, the performance anomaly was detected, and the faulty source was pinpointed to the hard disk on IO2. Because each indicator generates a boolean value at the end of a sampling period, for this experiment, the latency was $\Delta = 4 \times 4 - 13 = 3$ seconds, and A_{td} was 100%, as there was no false detection.

TABLE IV.: Testbed Information.

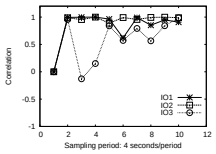
Server name	Type	CPU	Memory	HDD	Network Card
MDS	Metadata server	P4 CPU 2.53GHz	500MB	FUJITSU IDE 8GB 5400rpm	1Gbps
IO1	IO server	P4 CPU 2.40GHz	2026MB	SEAGATE SCSI 18.3GB 15000 rpm	1Gbps
IO2	IO server	P4 CPU 2.40GHz	1264MB	WDC IDE 40GB 7200rpm	1Gbps
IO3	IO server	P4 CPU 2.80GHz	1010MB	WDC SATA 250G 7200rpm	1Gbps



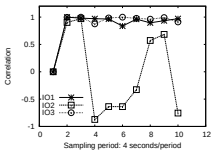
(a) External I/O request rate



(b) $corr(req, mem)$

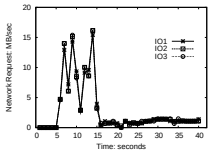


(c) $corr(req, interrupt)$

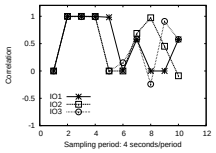


(d) $corr(iosize, svctm)$

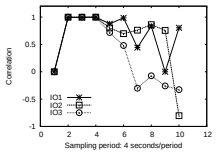
Figure 11: Disk delay faults were injected at the 13rd second, and the workload was 1GB sequential write on PVFS.



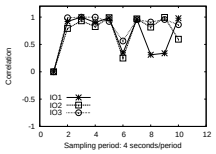
(a) External I/O request rate



(b) $corr(req, mem)$



(c) $corr(req, interrupt)$



(d) $corr(iosize, svctm)$

Figure 12: Network delay faults were injected at 15th second, and the workload was 1GB sequential write on PVFS.

B. Network delay faults

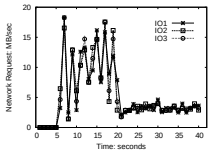
This set of experiments evaluated our performance anomaly detector in the presence of network delay faults which added extra delay at an I/O server for sending every request over the network. The delay was set to 50 ms in the following experiments. Figure 12 shows the results of 1GB sequential write tests on PVFS where the network delay faults were introduced at IO2.

In figure 12, network delay faults were injected at the 15th second at IO2. Our detector correctly detected the performance problem caused by the faults at the 5th sampling period. For this experiment, the detection latency was $\Delta = 5 \times 4 - 15 = 5$ seconds, and the true positive rate was $A_{td} = \frac{4}{4+1} = 0.8$, as the performance anomaly was not detected at the 4th sampling period.

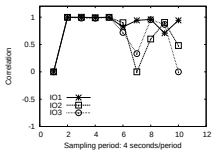
C. CPU overuse

This set of experiments evaluated our performance anomaly detector in the case of CPU overuse faults which make the available CPU resource at a low level. In the set of experiments, our fault injector occupied nearly 90% CPU resource in terms of the percentage of CPU time.

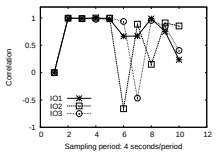
Figure 13 shows the results of the results of 1GB sequential write test on PVFS where CPU overuse faults were injected at the 19th second at IO2. Because I_{req} of IO2 generated a FALSE value at the 5th sampling period, our detector did not raise an alarm. However, our detector raised an alarm at the next sampling period, and correctly pinpointed CPU as the faulty source, as only the invariant for CPU of IO2 was broken. For this experiment, the detection latency was $\Delta = 6 \times 4 - 19 = 5$ seconds, and the true positive rate was $A_{td} = \frac{5}{5+1} \approx 0.83$.



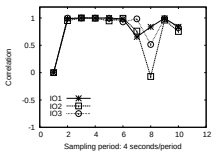
(a) External I/O request rate



(b) $corr(req, mem)$



(c) $corr(req, interrupt)$



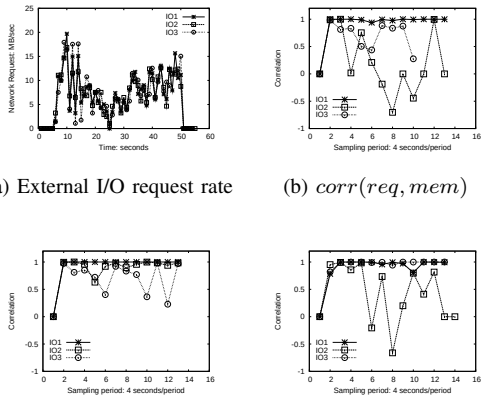
(d) $corr(iosize, svctm)$

Figure 13: CPU overuse faults were injected at 19th second, and the workload was 1GB sequential write on PVFS.

D. Memory overuse

This set of experiments evaluated our performance anomaly detector in the case of memory overuse faults which make the available memory resource at a low level. In the set of experiments, our fault injector occupied up to 90% memory resource.

Figure 14 shows the results of the results of 1GB sequential write test on PVFS where memory overuse faults were injected at the 12nd second at IO2. At the 7th sampling period, I_{req} of IO2 generated a TRUE value, and the invariant for memory of IO2 was broken, an alarm was raised. Because we gradually occupied system memory, every 100MB per second, it is reasonable that the negative impact of memory overuse faults cannot observed immediately. For this experiment, the detection latency was $\Delta = 7 \times 4 - 12 = 16$ seconds, and the true positive rate was $A_{td} = \frac{3}{3+4} \approx 0.43$.



(a) External I/O request rate (b) $corr(req, mem)$

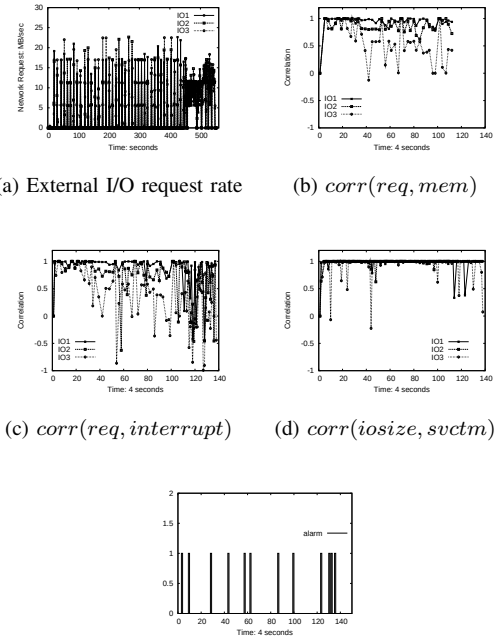
(c) $corr(req, interrupt)$ (d) $corr(iosize, svctm)$

Figure 14: Memory overuse faults were injected at 12nd second, the workload was 1GB sequential write on PVFS.

E. Benchmark Workload

The efficiency and accuracy of the detector have been demonstrated with synthetic workloads in different faulty situations; however, it is still necessary to examine the behaviors of the detector with realistic workloads to comprehensively evaluate it. In this section, a parallel I/O benchmark, BTIO, was adopted to evaluate the detector. BTIO is a tool contained in the NAS Parallel Benchmarks (NPB), and it is used to test the output capabilities of high-performance computing systems, especially distributed storage systems [27]. In our experiments, BTIO was complied with four processes, these processes work cooperatively to perform I/O operations on a dedicated storage system.

Figures 15 shows the results of BTIO test on PVFS in fault-free situation. For the test on PVFS, the invariants for memory and disk held at most time through the experiment; however, the invariant for CPU of all I/O servers did not hold well, especially, the invariant was frequently broken after the 115 sampling period in Figure 15c, meanwhile, a significant drop of req was also observed at the same time in Figure 15a. It is because BTIO performed read operations which requires frequently synchronization among all processes after the sampling period that more CPU time was occupied for synchronization. It is necessary to point out that there was no value generated by I_{mem} after the 116th sampling period, as there was few memory allocations after that. Because no fault was injected for the test, any alarm generated by the detector was marked as false detection. In the Figure 15e, there were a total of 12 alarms raised through the test, thus, the true positive rate was $A_{td} = \frac{126}{126+12} = 0.91$.



(a) External I/O request rate (b) $corr(req, mem)$

(c) $corr(req, interrupt)$ (d) $corr(iosize, svctm)$

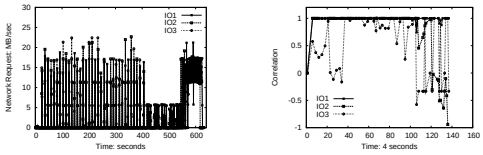
(e) Alarms

Figure 15: BTIO on PVFS.

1) Disk Delay Faults: This set of experiments evaluated the detector in the case of disk delay faults. The delay was set to 50 ms for the following experiments. Figures 16 shows the results of BTIO test on PVFS where the disk delay faults were introduced at IO2.

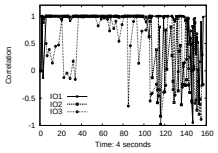
In Figure 16, disk delay faults were injected at the 410th second at IO2. The detector correctly detected the performance problem caused by the faults at the 109th sampling period. The detection latency was $\Delta = 109 \times 4 - 410 = 26$ seconds. The big latency was largely due to the discontinuous workloads generated by BTIO, as there was few incoming requests on IO3 between the 103rd and 109th sampling period. In Figure 16e, the detector raised four false alarms before the correct one, thus, the true positive rate was $A_{td} = \frac{111}{111+4} \approx 0.97$.

2) Network Delay Faults: This set of experiments evaluated the detector in the case of network delay faults. The delay was set to 50 ms for the following experiments. Figure 17 shows the results of BTIO test on PVFS where

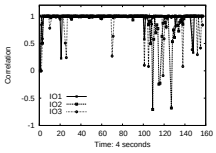


(a) External I/O request rate

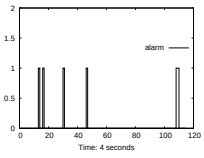
(b) $corr(req, mem)$



(c) $corr(req, interrupt)$

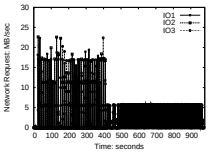


(d) $corr(iosize, svctm)$

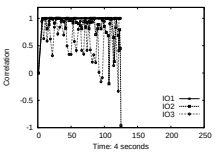


(e) Alarms

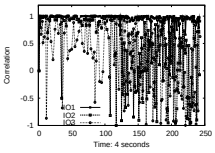
Figure 16: PVFS: Disk delay faults were injected at the 410th second at IO2.



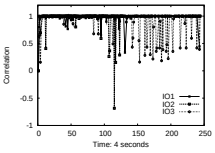
(a) External I/O request rate



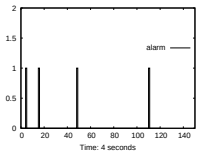
(b) $corr(req, mem)$



(c) $corr(req, interrupt)$



(d) $corr(iosize, svctm)$



(e) Alarms

Figure 17: PVFS: Network delay faults were injected at the 411th second at IO2.

the disk delay faults were introduced at IO2.

In Figure 17, network delay faults were injected at the 411th second at IO2. The detector correctly detected the performance problem caused by the faults at the 110th sampling period. The detection latency was $\Delta = 110 \times 4 - 411 = 29$ seconds. The big latency was largely due to the discontinuous workloads generated by BTIO, as there was no incoming requests on IO3 between the 100th and 108th sampling period. Meanwhile, the invariant for CPU of all I/O servers were more frequently broken after the 100th sampling period than the normal case, which can be regarded as a side effect of disk delay faults. In Figure 17e, the detector raised three false alarms before the correct one, thus, the true positive rate was $A_{td} = \frac{117}{117+3} \approx 0.98$.

F. Summary

Table V gives a summary of experiments with synthetic and BTIO workloads. For synthetic workloads, the detection latency is limited to two sampling periods (8 seconds), the average true positive rate is 84%, and there are no more than two false detections for most tests except the ones of memory overuse. The main reason for the poor performance of our detector in the experiments of memory overuse is that we gradually occupied system memory, our detector was insensitive to the small memory leak, as system performance was not significantly affected until a large portion of memory resource was leaked, thus our detector cannot detect immediately the faults of memory overuse.

Because the workloads generated by BTIO were not continuous, the average detection latency in the experiments with BTIO is larger than the one in the experiments with synthetic workloads. However, the average accuracy of 94% can be achieved by the detector for BTIO workloads.

VI. Conclusion

In this work, we presented a performance anomaly detector which is used to detect performance anomaly and accurately identify the faulty sources in an I/O server of cluster file systems. We concluded three invariants of an I/O server, which referred to the stable relationships between server workloads and resource statistics when the server works properly. By utilizing these invariants, a performance detector was developed, and the detector was evaluated with synthetic and BTIO workloads on PVFS file system in the presence of four different faulty situations. Our preliminary results demonstrated the efficiency and accuracy of the detector.

Acknowledgment

A preliminary version of this work was presented at the 3rd International Symposium on Parallel Architectures, Algorithms, and Programming (PAAP) [28].

This research is sponsored in part by National Science Foundation grants CNS-1102629, CCF-1102605, and CCF-1102624. This work is also partially supported by the SeaSky Scholar fund of the Dalian University of Technology. Any opinions, findings, and conclusions or recommendations expressed in this material are those of

TABLE V.: A Summary of Experiments with Synthetic Workloads.

Filesystem	Workload	Fault	Detection Latency	True positive rate	# of false detection
PVFS	1GB write	Disk delay	3 seconds	100%	0
		CPU overuse	5 seconds	83%	1
		Memory overuse	17 seconds	43%	4
		Network delay	5 seconds	80%	1
	1GB read	Disk delay	7 seconds	67%	2
		CPU overuse	7 seconds	80%	1
		Memory overuse	17 seconds	43%	4
		Network delay	6 seconds	80%	1
	BTIO	Disk delay	26 sec	97%	4
		Network delay	29 sec	98%	3

the author(s) and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] L. Cherkasova, K. M. Ozonat, N. Mi, J. Symons, and E. Smirni, "Anomaly? Application Change? or Workload Change? Towards Automated Detection of Application Performance Anomaly and Change," in *DSN '08: Proceedings of the International Conference on Dependable Systems and Networks*, June 2008, pp. 452–461.

[2] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan, "Black-Box Problem Diagnosis in Parallel File Systems," in *FAST '10: Proceedings of the 8th conference on File and Storage Technologies*, February 2010, pp. 57–70.

[3] X. Chen, J. Langston, and X. He, "An Adaptive I/O Load Distribution Scheme for Distributed Systems," in *PMEO-UCNS' 10: The 9th International Workshop on Performance Modeling, Evaluation, and Optimization of Ubiquitous Computing and Networked Systems in conjunction with IPDPS'10*, April 2010.

[4] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: an approach to designing fault-tolerant computing systems," *ACM Trans. Comput. Syst.*, vol. 1, no. 3, pp. 222–238, 1983.

[5] E. S. Buneci and D. A. Reed, "Analysis of Application Heartbeats: Learning Structural and Temporal Features in Time Series data for Identification of Performance Problems," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, November 2008, pp. 1–12.

[6] H.-L. Truong, P. Brunner, T. Fahringer, F. Nerieri, R. Samborski, B. Balis, M. Bubak, and K. Rozkwitalski, "K-WfGrid Distributed Monitoring and Performance Analysis Services for Workflows in the Grid," in *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, 2006, pp. 1–15.

[7] H. Chen, G. Jiang, and K. Yoshihira, "Failure Detection in Large-Scale Internet Services by Principal Subspace Mapping," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, no. 10, pp. 1308–1320, 2007.

[8] Z. Lan, Z. Zheng, and Y. Li, "Toward Automated Anomaly Identification in Large-Scale Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 174–187, 2009.

[9] S. Kavulya, R. Gandhi, and P. Narasimhan, "Gumshoe: Diagnosing Performance Problems in Replicated File-Systems," in *SRDS '08: Proceedings of the 2008 Symposium on Reliable Distributed Systems*, October 2008, pp. 137–146.

[10] Z. Guo, G. Jiang, H. Chen, and K. Yoshihira, "Tracking Probabilistic Correlation of Monitoring Data for Fault Detection in Complex Systems," in *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks*, June 2006, pp. 259–268.

[11] J. Gao, G. Jiang, H. Chen, and J. Han, "Modeling Probabilistic Measurement Correlations for Problem Determination in Large-Scale Distributed Systems," in *ICDCS '09: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, June 2009, pp. 623–630.

[12] B. Chun and A. Vahdat, "Workload and Failure Characterization on a Large-Scale Federated Testbed," *Intel Research Berkeley Technical Report IRB-TR-03-040*, November 2003.

[13] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, 2003, p. 1.

[14] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang, "Failure Data Analysis of a Large-Scale Heterogeneous Server Environment," in *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, June 2004.

[15] P. Yalagandula, S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Beyond Availability: Towards a Deeper Understanding of Machine Failure Characteristics in Large Distributed Systems," in *First Workshop on Real Large Distributed Systems (WORLDS)*, December 2004.

[16] B. Schroeder and G. A. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," in *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks*, June 2006, pp. 249–258.

[17] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky, "Are disks the Dominant Contributor for Storage Failures? a Comprehensive Study of Storage Subsystem Failure Characteristics," in *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, February 2008, pp. 1–15.

[18] L. N. Bairavasundaram, G. R. Goodson, B. Schroeder, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dussea, "An Analysis of Data Corruption in the Storage Stack," in *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, February 2008, pp. 233–238.

[19] W. Jiang, C. Hu, S. Pasupathy, A. Kanevsky, Z. Li, and Y. Zhou, "Understanding Customer Problem Troubleshooting from Storage System Logs," in *FAST '09: Proceedings of the 7th conference on File and storage technologies*, February 2009, pp. 43–56.

[20] G. Jiang, H. Chen, and K. Yoshihira, "Discovering Likely Invariants of Distributed Transaction Systems for Autonomic System Management," in *ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, 2006, pp. 199–208.

[21] M. Placek and R. Buyya, "A Taxonomy of Distributed Storage Systems," www.cloudbus.org/reports/DistributedStorageTaxonomy.pdf.

[22] X. Chen, J. Langston, and X. He, "Design and Evaluation of a User-Oriented Availability Benchmark for Distributed File Systems," in *PDCS '09: Proceedings of the 21st IASTED International Conference on Parallel and Distributed Computing and Systems*, November 2009.
[23] "PVFS," March 2009, http://www.pvfs.org/.
[24] "Lustre," July 2008, http://en.wikipedia.org/wiki/Lustre_(file_system).
[25] D. P. Bovet and M. Cesati, *"Understanding the Linux Kernel: From I/O Ports to Process Management, 3nd"*. O'Reilly Media, Inc., 2006, iSBN: 0-596-00565-2.
[26] Y. Hu, A. Nanda, and Q. Yang, "Measurement, Analysis and Performance Improvement of the Apache Web Server," in *Proceedings of the IEEE International Performance, Computing, and Communications Conference*, 1999, pp. 261–267.
[27] P. Wong and R. F. V. der Wijngaart, "NAS Parallel Benchmarks I/O Version 2.4," 2003, http://www.nas.nasa.gov/News/Techreports/2003/PDF/nas-03-002.pdf.
[28] X. Chen, X. He, H. Guo, and Y. Wang, "An Online Performance Anomaly Detector in Cluster File Systems," in *PAAP '10: Proceedings of the 3rd International Symposium on Parallel Architectures, Algorithms, and Programming*, December 2010.

Xin Chen received both the MS and PhD degrees in electrical engineering from Tennessee Technological University, USA, in 2007 and 2010, respectively, and his BS degree in Mechanical Engineering from Shanghai University, China in 2003. He is currently a system engineer at Dell Inc., Austin, Texas, USA. His research interests include computer architecture and storage systems.

Xubin He received the PhD degree in electrical engineering from University of Rhode Island, USA, in 2002 and both the BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1995 and 1997, respectively. He is currently an associate professor in the Department of Electrical and Computer Engineering at Virginia Commonwealth University. His research interests include computer architecture, storage systems, virtualization, and high availability computing. He received the Ralph E. Powe Junior Faculty Enhancement Award in 2004 and the TTU Chapter Sigma Xi Research Award in 2005 and 2010. He is a senior member of the IEEE and a member of the IEEE Computer Society.

He Guo received both the BS and MS degrees in computer science from Jilin University, China, in 1982 and Dalian University of Technology, China, in 1989, respectively. He is currently a professor in the Software School at Dalian University of Technology. His research interests include computer architecture, virtualization, parallel computing and computer vision. He is a member of the IEEE Computer Society.

Yuxin Wang received both the BS and MS degrees in computer science from Dalian University of Technology, China, in 1997 and 2000, respectively. He is currently a lecturer in the School of Computer Science and Technology at Dalian University of Technology. His research interests include computer architecture, virtualization and pattern recognition.