An Online Performance Anomaly Detector in Cluster File 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 38505, USA*
[2]*Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA 23284, 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*
[*]*xchen21@tntech.edu,* [†]*xhe2@vcu.edu,* [‡]*guohe@dlut.edu.cn,*[§]*wyx@dlut.edu.cn*

***Abstract*—Performance problems, which can stem from different system components, such as network, memory, and storage devices, are difficult to diagnose and isolate in a cluster file system. In this paper, we present an online performance anomaly detector which is able to efficiently detect performance anomaly and accurately identify the faulty sources in a system node of a cluster file 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 preliminary experimental results demonstrate the efficiency and accuracy of the proposed performance anomaly detector.**

***Keywords*-performance anomaly detector; cluster file system;**

I. Introduction

Performance is critical in the study of file systems. Synthetic workloads or file system benchmarks are created to examine the behaviors of file systems. Although they are very useful in the initial stage of the design and development of a file system, it is insufficient for using them to analyze or resolve one common problem called performance anomaly in file systems [1], [2]. By performance anomaly we mean that the current observed system behavior is 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 file systems. However, it remains a challenging task to efficiently detect performance anomaly and accurately identify the faulty sources, particularly in cluster file systems.

Cluster file systems like PVFS [3] 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 in the systems, instead, the performance is often limited by the capability of the slowest ones [4], [2]. So, if there exists performance anomaly in some node of a cluster file 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 cluster file 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 [5], it is more difficult to detect the existence of performance anomaly, and even more difficult to identify the source of the anomaly, since both dynamic workload change and many uncertain factors such as caching and scheduling can perplex our 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 [6], [7]. 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 [8], [9], [2], however, their usages are limited to the generality of their models.

In this work, we target the runtime diagnosis of performance anomaly in cluster file systems which may consist of heterogeneous computer nodes and experience dynamic changed workloads. Our approach is self-diagnosis based, which exploits some invariants exist in a computer node of a cluster file system to detect the performance anomaly and identify faulty sources of that node. Such invariants refer to the stable relationships between workloads and system resource statistics in faulty-free situations. An online performance anomaly detector was developed based on these invariants, and our preliminary experimental results demonstrate the efficiency and accuracy of the detector.

The rest of the paper is organized as follows: Section 2 gives a brief discussion of related works. In section 3, we describe our methodology for performance anomaly detection and identification, and present the design of our performance anomaly detector in section 4. Section 5 describes our experiments and lists experimental results. Finally, we conclude the paper in section 6.

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 [6], [7]. 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 [10] and Lan et al [9] 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 [8] 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 [11] and Gao et al [12] 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

In our opinion, 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 [13]. Meanwhile, if a resource request from a process cannot be satisfied immediately, the kernel also forces the process sleep. Thus, our option of utilizing invariants in a computer node to detect performance anomaly drive us to explore the relationships between workloads and system resource statistics. Here, invariants refer to the stable relationships between workloads and system resource statistics in faulty-free situations. By studying the trace data collected from our previous studies on distributed storage systems [4], [14], we concluded three invariants as follows.

[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 is 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 helps us to identify the performance problems originated from memory.

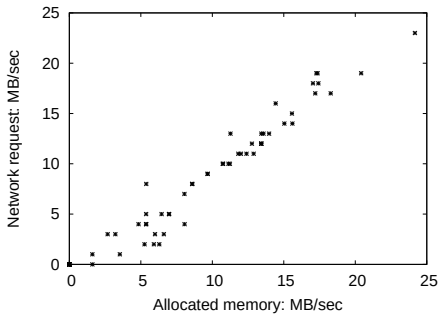


Figure 1: The relation between the total size of I/O requests over network per second and the amount of allocated memory per second. Data is from a trace of 40 seconds I/O activities on a computer node.

[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 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 [15]. Once the CPU resource is insufficient for the distributed file system process, fewer I/O related interrupts are generated, and the proportional relationship 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.

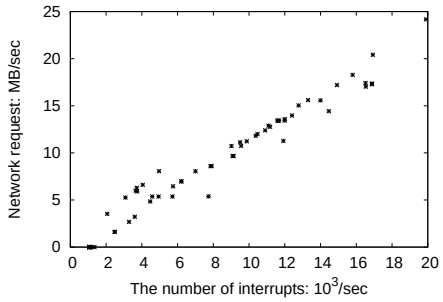


Figure 2: The relation between the total size of I/O requests over network per second and the number of interrupts per second. Data is from a trace of 40 seconds I/O activities on a computer node.

[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.

A. Indicators of Performance Anomaly

Although one or more of our 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.

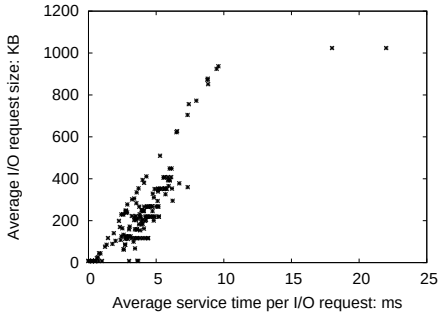


Figure 3: The relation between average I/O request size and average service time per I/O request. Data is from a trace of 250 seconds I/O activities on a computer node.

To compensate the drawback of our invariants, an indicator $I_{req}(n)$ is adopted to detect the performance anomaly on an I/O server at the nth sampling period. Formula 1 gives the definition of $I_{req}(n)$, where req_{n-1} denotes the average total size of I/O requests from clients at the $(n-1)$th sampling period, req_n denotes the average total size of I/O requests from clients 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, we cannot use only $I_{req}(n)$ to detect performance anomaly on an I/O server, because non-faulty I/O servers also observer the degradation of receiving request rate [2]. $I_{req}(n)$ should be combined with the indicators of our invariants to detect performance anomaly.

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

B. Indicators of Faulty Sources

Because the invariants we discussed refer to a proportional relationship between two metrics, in order to use them in practice, such a proportional relationship needs to be quantified. The correlation $corr(x, y)$ provides us a good measurement of proportional relationships between two variables: x and y. Formula 2 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.

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

Thus, based on formula 2, three indicators I_{mem}, I_{cpu}, and I_{disk} are defined to test our invariants by formula 3, 4, and 5, 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 I lists the parameters used in these formulas.

$$I_{cpu} = \begin{cases} \text{FALSE, if } corr(req, interrupt) < 0 \\ \text{TRUE, if } corr(req, interrupt) \geq 0 \end{cases} \quad (3)$$

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

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

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 6, 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 6, $I^n_{network}$ 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.

$$\begin{cases} I^n_{network} = I^n_{disk} \wedge I^n_{mem} \wedge I^n_{cpu} \wedge I^n_{req},\ n \in \mathbb{N} \\ I'_{network} = I^1_{network} \wedge I^2_{network} \wedge \cdots \wedge I^n_{network},\ n \in \mathbb{N} \end{cases} \quad (6)$$

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 4 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.

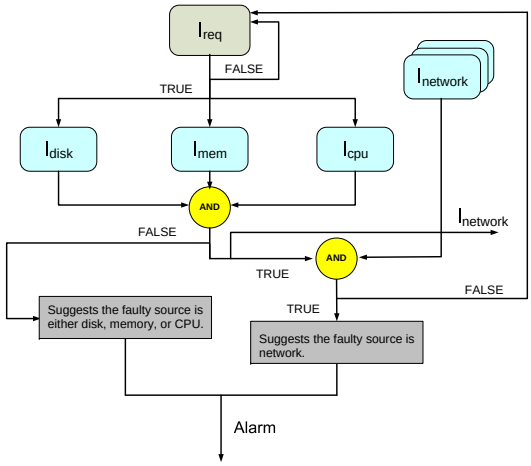


Figure 4: The working flow of the online performance anomaly detector.

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 II. 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

Table I: Symbols in formula 3, 4, and 5.

Parameter	Description
req	the total size of incoming I/O requests from clients 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 II: Testbed Information.

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

indicators were evaluated at the end of the period; we set α to 50% for I_{req}.

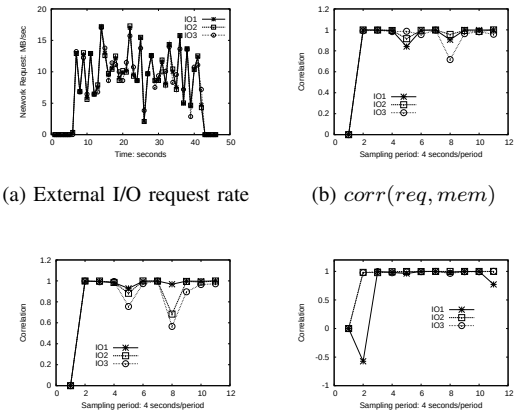
In order to measure the efficiency and accuracy of our detector, two metrics are defined: the detection latency and the true positive rate. The former measures how long our detector may take to detect the existence of performance anomaly after the injection of performance faults, and the latter measures the accuracy of our detector in terms of the percentage of correct alarms. Formula 7 and 8 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.

$$\Delta = T_d - T_i \tag{7}$$

$$A_{td} = \frac{N_{td}}{N_{td} + N_{fd}} \tag{8}$$

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 5 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 5d, 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.



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

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

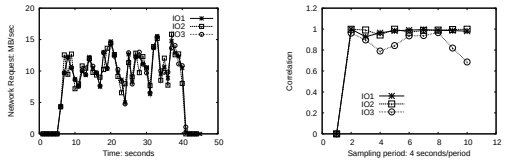
Figure 5: 1GB sequential write on PVFS.

The results of 1GB sequential read tests on PVFS in fault-free situation are shown in figure 6. As similar as in figure 5, 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 6, 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-E.

A. Disk delay faults

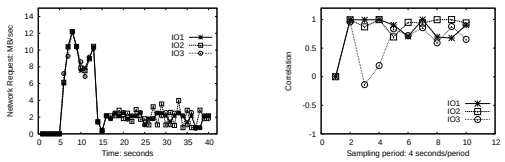
This set of experiments evaluated our performance anomaly detector in the case of disk delay faults which



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

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

Figure 6: 1GB sequential read on PVFS.



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

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

Figure 7: Disk delay faults were injected at the 13rd second, and the workload was 1GB sequential write on PVFS.

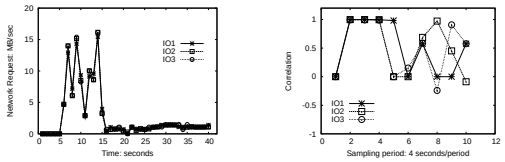
do not fail any I/O request but introduce extra I/O request processing time in a hard disk driver. The delay was set to 50 ms for the following experiments. Figure 7 shows the results of 1GB sequential write test on PVFS where the disk delay faults were introduced at the 4th sampling period (13rd – 16th second) at IO2.

In figure 7, 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 7a, 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.

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 8 shows the results of 1GB sequential write tests on PVFS where the network delay faults were introduced at IO2.

In figure 8, 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.



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

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

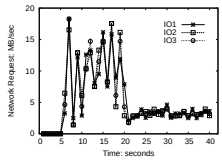
Figure 8: Network delay faults were injected at 15th second, and the workload was 1GB sequential write on PVFS.

C. CPU overuse

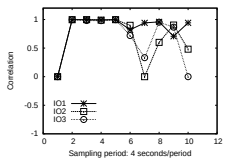
This set of experiments evaluated our performance anomaly detector in the case of CPU overuse faults which make the available CPU resource at a low level. In the set of experiments, our fault injector occupied nearly 90% CPU resource in terms of the percentage of CPU time.

Figure 9 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

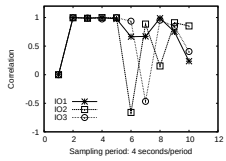
at the next sampling period, and correctly pinpointed CPU as the faulty source, as only the invariant for CPU of IO2 was broken. For this experiment, the detection latency was $\Delta = 6 \times 4 - 19 = 5$ seconds, and the true positive rate was $A_{td} = \frac{5}{5+1} \approx 0.83$.



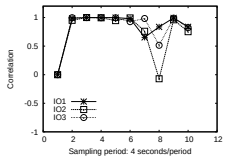
(a) External I/O request rate



(b) $corr(req, mem)$



(c) $corr(req, interrupt)$



(d) $corr(iosize, svctm)$

Figure 9: CPU overuse faults were injected at 19th second, and the workload was 1GB sequential write on PVFS.

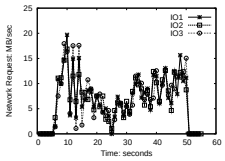
D. Memory overuse

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.

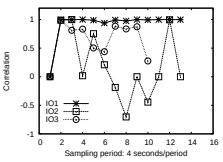
Figure 10 shows the results of the results of 1GB sequential write test on PVFS where memory overuse faults were injected at the 12nd second at IO2. At the 7th sampling period, I_{req} of IO2 generated a TRUE value, and the invariant for memory of IO2 was broken, an alarm was raised. Because we gradually 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 $A_{td} = \frac{3}{3+4} \approx 0.43$.

E. Summary

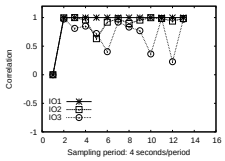
Table III gives a summary of experiments with synthetic workloads. In the table, the detection latency is limited to two sampling periods (8 seconds), the average true positive rate is 84%, and there are no more than two false detections for most tests except the ones of memory overuse. The main reason 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



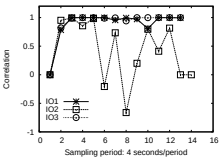
(a) External I/O request rate



(b) $corr(req, mem)$



(c) $corr(req, interrupt)$



(d) $corr(iosize, svctm)$

Figure 10: Memory overuse faults were injected at 12nd second, the workload was 1GB sequential write on PVFS.

affected until a large portion of memory resource was leaked, thus our detector cannot detect immediately the faults of memory overuse.

VI. Conclusions

In this work, we presented an online 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, an online performance detector was developed, and the detector was evaluated with synthetic workloads on PVFS in the presence of four different faulty situations. Our preliminary results demonstrated the efficiency and accuracy of our detector.

Acknowledgments

This research is sponsored in part by National Science Foundation grants CNS-0720617, CCF-0937850, and CCF-0937799. 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 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.

Table III: 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

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

[3] "PVFS," March 2009, http://www.pvfs.org/.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] X. Chen, J. Warren, F. Han, and X. He, "Characterizing the dependability of distributed storage systems using a two-layer hidden markov model-based approach," in *Proceedings of the International Conference on Networking, Architecture, and Storage (NAS)*, 2010.

[15] 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.