Proceedings of the 5th Workshop on Hot Topics in System Dependability (HotDep '09).
Lisbon, Portugal. June 2009.

System-Call Based Problem Diagnosis for PVFS

Michael P. Kasick, Keith A. Bare, Eugene E. Marinelli III, Jiaqi Tan, Rajeev Gandhi, Priya Narasimhan
Electrical & Computer Engineering Department; Carnegie Mellon University; Pittsburgh, PA 15213-3890
Email: {mkasick, kbare, emarinel, jiaqit, rgandhi, priyan}@andrew.cmu.edu

Abstract

We present a syscall-based approach to automatically diagnose performance problems, server-to-client propagated errors, and server crash/hang problems in PVFS. Our approach compares the statistical and semantic attributes of syscalls across PVFS servers in order to diagnose the culprit server, under these problems, for different file-system benchmarks—dd, PostMark and IOzone—in a PVFS cluster.

1. Introduction

The Parallel Virtual File System (PVFS) [1] is an open-source, parallel file-system that provides high-performance computing (HPC) applications with high-speed data read/write access to files in a cluster of commodity computers. PVFS is designed as a client-server architecture, with many clients communicating with multiple I/O servers and one or more metadata servers. To facilitate parallel access to a file, PVFS distributes (or "stripes") that file across multiple disks located on physically distinct I/O servers.

Problem diagnosis is important in long-running jobs in HPC environments. Often the effects of problems can be magnified due to computations exhibiting long durations and being performed at large scales. Server-to-client propagated errors and server crash/hang failures are two other relevant categories of PVFS problems. Current diagnosis involves the manual analysis of client/server debug logs that record PVFS operations; this logging has a high runtime overhead and requires code-level (white-box) PVFS instrumentation.

On the other hand, syscall tracing does not require any modification of either the traced client application or PVFS, making it a black-box diagnosis strategy. Syscall instrumentation of the PVFS server and client processes yields *statistical* data about disk/network I/O transfer times that then enable us to detect performance anomalies. Syscall instrumentation of the client application yields *semantic* data about hung or failed requests, which we then trace back to misconfiguration or resource-exhaustion.

Concretely, our contributions are: (i) a new syscall-based approach to diagnose problems automatically and transparently in parallel file-systems, such as PVFS, (ii) a statistical diagnosis algorithm that correlates syscall service-times across PVFS servers, to localize the culprit server, and (iii) a semantic diagnosis algorithm that correlates errors returned by syscalls at the PVFS client and servers, to diagnose non-performance problems.

2. Problem Statement

Our research seeks to explore to what extent syscall-based instrumentation is useful in diagnosing a variety of file system problems, including those that occur in or are motivated by production deployments (see § 3).

We aim for our approach to be: (i) transparent, i.e., no modifications to the PVFS applications, and independent of PVFS's operation; (ii) able to differentiate between anomalous and legitimate behavioral changes (e.g., workload shifts); (iii) able to diagnose the culprit server under performance problems, misconfigurations, and resource exhaustions that cause degraded or halted PVFS operation. Fine-grained diagnosis, which would trace the bug to culprit lines of PVFS source-code, is outside our current scope.

We assume that a majority of the PVFS servers exhibit fault-free behavior. We assume that the physical clocks on the cluster's nodes are synchronized so that their time-stamped data can be temporally correlated.

Hypotheses: Based on PVFS's design, we have specific expectations of its behavior. For performance problems, we expect non-faulty I/O servers to exhibit similar wall-clock service times for read/write syscalls, while we expect a faulty I/O server to exhibit longer service times for the same syscalls. Thus, the statistical comparison of read/write syscall durations, across PVFS servers, should identify an anomalous server.

For server-to-client propagated errors, we expect a server's host file-system's errors to propagate, through

the server, and manifest as an error at a client application's syscall. By observing and correlating such errors at both the client application and the servers, we should be able to identify the culprit server. For server crash/hang failures, we expect that client application syscalls will either return with a connection error (server crash) or never terminate (server hang). Thus, by waiting for a connection error or explicit time-out, we should be able to identify when a crash/hang has occurred and then trace it back to the culprit server.

3. Anecdotal Motivation

The faults that we study in our work are motivated by the PVFS developers' anecdotal experience [2] of problems faced/reported in various production PVFS deployments, one of which is Argonne National Laboratory's 557 TFlop Blue Gene/P (BG/P) PVFS cluster.

Anomalous disk behavior motivates two of our performance problems. RAID controllers that inadvertently create disk contention while proactively searching for media defects [3], [4] motivate our *disk-busy* problem. The accidental launch of an interfering rogue process such as a second, colocated file server (e.g., PVFS and GPFS) [2] motivates our *disk-hog* problem.

Network problems primarily manifest in packet-loss errors, which are the "most frustrating" [sic] to diagnose [2]. They often result from faulty switch ports that enter a degraded state and send CRC-failing packets, spreading poor performance throughout the network. They also result from overloaded switches that "just can't keep up" [sic], in which case diagnostic tests of individual links might exhibit no errors, yet problems will manifest while PVFS is running [2].

Non-performance problems (e.g., misconfigurations that leads to server crashes) are somewhat easier to detect as they typically alter the behavior of the system in an observable fashion, e.g., by the application returning errors or indefinitely hanging during an operation, potentially triggering an application-level timeout. While easier to detect than performance problems, it is still difficult to diagnose the faulty server.

For instance, PVFS bug ticket #50 documents a scenario where a single I/O server was accidentally misconfigured with an unintentionally low resource limit [5]. The server, at its configured open-file limit, reacted to an attempt to open a new client socket by killing the message-handling thread which hung all subsequent requests. Our study of this problem reveals a more subtle manifestation: if the open-file limit is reached when opening a new storage file (instead of a new socket) the `open` syscall returns an EMFILE `errno` that is propagated to the client application and returned unintuitively from a `read` or `write` syscall.

Finally, intermittent disk availability problems, such as Fibre Channel disks that are temporarily unplugged, can result in a failure mode where the host file-system switches over to read-only operation [2], [6]. In our study of this problem, PVFS returns either an EROFS `errno`, or suspends a server thread, depending on the operations that are performed after this event.

4. Instrumentation

We have developed a tool, `syscap`, that uses `ptrace` (a user-space Linux API for syscall interception and modification, *without* changes to the monitored process) to trace syscalls and signals. Each local process' traced events are written, in the order of event completion, as a record to a syscall-event log (`sclog`). Record fields include a record number, a sequence number (identifying events in order of start), timestamp (at the start of the event), light-weight process ID, syscall/signal number, syscall register arguments, syscall result, and syscall wall-clock service time.

`syscap` also produces a file-descriptor update-log (`fdlog`) with records that describe updates to traced processes' file-descriptor tables due to syscalls (e.g., `open`, `close`) that change the open-file table. The target file name is provided by the `/proc/pid/fd/#` symlink. For anonymous "files" (e.g., network sockets), a special identifier captures other characteristics, e.g., TCP-socket IP address and port number. On observing the creation of a file-descriptor, we automatically track all syscalls that subsequently use the file-descriptor, and extract the service times of all associated read and write syscalls.

Diagnosing performance problems. The metrics of interest are: (i) disk-read service time (`dread`), (ii) disk-write service time (`dwrite`), (iii) server's network-read time (`nsread`), and (iv) client's network-read time (`ncread`). `dread` and `dwrite` are the values of the wall-clock service times for read and write syscalls, respectively, on I/O server file-objects. `nsread` and `ncread` represent the amount of time that it takes for the server (client) to read a single PVFS request (response) over the network.

Diagnosing propagated errors. We also seek to study errors that are returned from PVFS server syscalls to an server's host file-system, which are then propagated to clients, and finally appearing as return values from client application read/write syscalls. The metrics of interest for diagnosing propagated errors

are: (i) the errnos of failed syscalls, and (ii) timestamps of failed syscalls.

Diagnosing crash/hang faults. We also seek to study cases where a PVFS server either crashes, closing all client connections, or stops responding to requests, leaving clients in a hung state. Although execution halts in either case, it is still difficult to manually diagnose the faulty server. The metrics of interest for diagnosing crash/hang faults are: (i) service time of application's PVFS I/O syscalls, and (ii) most recent client syscall to each server socket.

5. Experimental Set-up

We perform our experiments on a cluster of AMD Opteron 1220 machines, each with 4 GB RAM, two Seagate Barracuda 7200.10 320 GB disks (one dedicated for PVFS storage), and a Broadcom NetXtreme BCM5721 Gigabit Ethernet controller. Each node runs Debian GNU/Linux 4.0 (etch) with Linux kernel 2.6.18 and PVFS 2.8.0. Server nodes use the PVFS Direct I/O storage method[1] with a 4 MB Flow buffer size[2]. Our PVFS experiments consist of 10 combined I/O and metadata servers and 10 clients.

Our study of performance problems involves the workload running for 120 seconds in fault-free mode, the fault injected for 300 seconds and then deactivated. The experiment continues to the completion of the benchmark, typically taking 600 seconds in the fault-free case. Our study of propagated errors and crash/hang faults involves the fault persisting for the entire experiment, which runs for shorter durations (approx 60 seconds), long enough for the workload to activate the fault.

Workloads. Our first two workloads, ddw and ddr, consist of the same benchmark, dd, either writing zeros (from /dev/zero) to a client-specific temporary file in PVFS, or reading the contents of a previously written client-specific temporary file and writing the output to /dev/null. dd models HPC scientific workloads with constant data-write rates.

Our next two workloads, iozonew and iozoner, comprise the same common file-system benchmark, IOzone v3.283. We run iozonew in write/rewrite mode and iozoner in read/reread mode. IOzone is a large-file I/O-heavy benchmark with few metadata operations, with an fsync and a workload change half-way through the benchmark. Our fifth benchmark is PostMark v1.51, a metadata-server heavy workload with small file writes (all writes < 64 kB; thus, writes occur only on a single server per file).

For the ddw workload, we use a 17 GB file with a record size of 40 MB. Write-record sizes were chosen so that 4 MB of data is sent to each server in a single bulk transfer, which is necessary for good PVFS performance. For ddr, we use a 27 GB file with a record-size of 40 MB. For iozonew, we use a 8 GB file with a record-size of 16 MB (the maximum supported by IOzone). For iozoner, we use a 8 GB file with a record-size of 16 MB. For postmark, we use the default configuration with 9,000 transactions.

Injecting performance problems. The six faults we inject that manifest as performance problems are:

- *disk-hog*—A dd process that reads 256 MB blocks from an unused partition on one of our storage disks.
- *disk-busy*—An sgm_dd process that issues low-level SCSI I/O commands to read 1 MB blocks from the same unused partition on our storage disk.
- *write-network-hog*—A third-party node opens a TCP connection to a listening port on one of the PVFS servers and sends zeros to it.
- *read-network-hog*—One of the PVFS servers opens a connection and sends zeros to the third-party node.
- *receive-pktloss*—Server receive-packet-loss. A netfilter firewall rule probabilistically drops packets received at one of the servers with probability 5%.
- *send-pktloss*—Server send-packet-loss. A firewall rule on all clients probabilistically drops packets incoming from a single servers with probability 5%.

Injecting propagated errors. The three faults we inject that manifest as propagated errors are:

- *err-inodes*—Insufficient inodes for storage space. An ENOSPC errno is returned by the server's host file-system and propagated to the client application through a write syscall, potentially misleading the client into believing that PVFS ran out of metadata structures. To inject this, we set one of the server's storage partition to have low number of inodes (2048) for its size (8 MB).
- *err-files*—Insufficient maximum open files (rlimit). An EMFILE errno is returned by the server to the application, unusually as the result of a read or write call, misleading the client into believing its resource limit (and not the server's) is misconfigured. To inject this, we execute one of the PVFS servers as an unprivileged user with RLIMIT_NOFILE set to 50.
- *err-remount*—Emergency remount read-only. This occurs when a PVFS server experiences an error due to the underlying storage device or file-system corruption. To inject this, at 15 sec-

1. Ensures metrics reflect disk service times instead of cache hits.
2. Allows larger bulk transfers for more efficient disk usage.

onds into the experiment, we write a "u" to /proc/sysrq-trigger, forcing the server to remount as read-only. We run a custom workload for this experiment that repeatedly opens a new file and writes a single null byte, then waits half a second. This ensures that the next PVFS operation after the fault-injection is an open (not a write) call.

Injecting crash/hang faults. The four faults we inject that manifest as crash/hang failures are:

- *err-space*—Insufficient storage partition space. The server's host file-system returns an ENOSPC errno for a write syscall. The server reacts by killing the message-handling thread, hanging all subsequent client requests. To inject this, we select a server and set its storage-partition size lower (256 MB).
- *err-fsize*—Insufficient maximum file size (rlimit). Results in a server write call returning an EBIG errno; then the process is signaled SIGXFSZ leading to a server crash. To inject this, we execute one of the server processes as an unprivileged user with the maximum file-size set to 128 MB.
- *err-vmsize*—Insufficient process maximum virtual-memory size (rlimit). Results in a server memory-allocation syscall (e.g., mmap) returning an ENOMEM errno; then the process is signaled SIGABRT leading to a server crash. To inject this, we execute one of the server processes as an unprivileged user with the maximum virtual-memory size to 300,000 kB.
- *err-remount*–Emergency remount read-only. Same as propagated error but uses a ddw workload with unbounded file-size. Results in server hang.

6. Statistical Syscall-Based Diagnosis

This diagnosis algorithm relies on our hypothesis that fault-free I/O servers have similar average behavior. However, even under fault-free conditions, servers can behave differently (even small hardware differences can cause some of the servers to be saturated). We account for such heterogeneity through a *fault-free training phase*. The underlying hypothesis is that while a saturated server's service time would likely differ from those of non-saturated servers under fault-free conditions, the deviation of a faulty server would be more pronounced.

For each syscall of interest, we generate a time-series of average syscall service times at each server by dividing the sum syscall duration by the number of syscalls at 1-second intervals. Our algorithm compares the time-series of syscall service times at all of the servers to detect any anomalous time-series (and thence, the associated anomalous server). Every second, we compute a representative value (currently, we use the median of the syscall service times across all the non-faulty/non-saturated servers) of the syscall service times for the non-faulty nodes. A server is flagged as anomalous if its syscall service time differs by more than a predetermined threshold from this representative value.

In the training phase, we determine the maximum deviation of a server's syscall service times from the representative (median) value under fault-free conditions. This phase also determines which servers are likely to be saturated under fault-free conditions. We currently use the ddw and ddr workloads only under fault-free conditions to determine the threshold values. These workloads place the highest throughput demands on server disks, and thus are most likely to saturate some servers and produce maximum deviation. For each server, we detect the maximum deviation of its service times from the median value and use the maximum deviation to choose our threshold value for that server. For actual deployment, a PVFS administrator would first run the ddr and ddw workload under fault-free conditions to determine the threshold values for each server.

7. Semantic Syscall-Based Diagnosis

Propagated errors. Our strategy consists of two phases: (i) a training phase to identify (and ignore) errnos that result from "failed" syscalls in normal PVFS operation, and (ii) a diagnosis phase to identify unexpected failed syscalls with propagated errors.

For training, we invoke the pvfs2-ping utility on a single client immediately after launching all of the PVFS servers. This exercises the code paths involved in a complete client session. As the utility executes, we examine the errnos returned by "failed" syscalls on the servers and flag these errnos as normal so that they can be safely considered as a part of normal PVFS operation. For diagnosis, if a server syscall returns an errno that we did not previously flag as normal, we examine whether the same errno is returned to the client application during a 3-second window around the server call's timestamp. If the errno is indeed propagated to the client, we declare the server where the errno originated as the culprit.

Crash/hang faults. Our examination of PVFS's behavior leads to three observations: (i) application syscalls that fail with connection error indicate that one or more PVFS servers has crashed, (ii) syscalls that never return indicate that one or more servers

has hung, and (iii) the crashed/hung servers can be identified based on whether or not the client daemon has received a response from the server for its most recent request. Thus, we first examine the client application's syscalls to find I/O syscalls that either return ECONNREFUSED (server crash) or exceed a timeout threshold of 30 seconds[3] (server hang). Second, we examine the PVFS client daemon's I/O syscalls to server sockets, keeping track of the syscall that was most recently performed in communication with each server, until the time of the application-syscall error or timeout. Based on these observations, we expect that the last syscall completed with the faulty server will differ from that completed with the non-faulty servers.

8. Results

Table 1 shows the performance (true- and false-positive rates[4] of our statistical algorithm for diagnosing performance faults using different syscalls for the different workloads. For each fault, we only show those syscalls causing a non-zero true/false positive.

The `dread` and `ncread` syscalls, which are not used by the write-intensive (`ddw` and `iozonew`) workloads, do not help with diagnosis for those workloads. Thus, the left half of the table omits the `ddw` and `iozonew` workloads. Similarly, the `dwrite` and `nsread` syscalls, which are not used by the read-intensive (`ddr` and `iozoner`) workloads, do not help with diagnosis for those workloads. Thus, the right half of the table omits the `ddr` and `iozoner` workloads. We include the `postmark` workload in both halves of the table since it includes read as well as write syscalls.

Our low false-positive rates are an artifact of our threshold selection (which minimizes only the false-positive rate). An ROC-based approach to threshold selection would likely increase the false-positive (as well as the true-positive) rate. Different syscalls are useful for diagnosing different problems. For instance, `dread` and `dwrite` are useful for diagnosing disk-related problems while `nsread` and `ncread` are useful for diagnosing network-related problems. While each syscall diagnoses only a subset of the problems, a combination of syscalls can effectively diagnose many performance problems.

For propagated errors and crash/hang faults, our diagnosis algorithms successfully diagnosed the faulty server with no false positives.

3. Must be longer than the time required to flush client flow buffers to avoid misdiagnosis in the event of a server crash. 15 seconds or more is sufficient for our experiments.

4. TP is the fraction of experiments where all faulty servers are correctly diagnosed as faulty, FP is the fraction where at least one non-faulty server is misdiagnosed as faulty.

	Overhead for Workload				
Inst.	ddr	ddw	iozoner	iozonew	postmark
syscap	0.2%	0.6%	−0.4%	0.7%	64.4%
strace	−0.1%	−0.8%	−3.3%	0.0%	138.8%
sysstat	0.7%	0.4%	−0.9%	0.4%	5.4%

Table 2. Instrumentation overhead: Increase in run-time w.r.t. non-instrumented workload.

Overheads. Table 2 reports overheads for three different kinds of black-box instrumentation[5] for our five workloads. They are calculated as the increase in mean workload runtime over uninstrumented counterparts.

Since four of our five workloads are I/O-heavy, with large bulk transfers, relatively few syscalls are made for the amount of data transferred. Since network and disk transfers consume the majority of time in these I/O operations, the added overhead (< 1%) is negligible.

In contrast, the metadata-heavy `postmark` workload has many small data transfers or metadata operations (create, remove, etc.) on many small files. Since the time to issue the syscall takes as long as (if not longer than) the time to carry out the requested operation, syscall instrumentation has a significant overhead (64% and 139% for `syscap` and `strace`, respectively). Because sysstat-based instrumentation does not alter the operation of syscalls, its overheads are more modest, making it more appropriate for metadata-heavy workloads.

9. Related Work

Past work on failure diagnosis in distributed systems has focused mainly on Internet services [7], [8], with some work examining debugging performance problems in high-performance computing environments [9]. Our work is different in targeting HPC file-systems instead of the deployed workload application.

Tracing and instrumentation generate system views that are useful for failure diagnosis. File system-specific tracing mechanisms include Stardust [10], which traces causal request flows through a distributed storage system, and TraceFS [11], which uses a thin file-system interpositioned between the Linux VFS layer and the underlying file-system to provide operation traces at multiple granularities. Performance tools for HPC environments include TAU [12] and Paradyn Parallel Performance Tools [13].

Shen [14] and OSprof [15] both use syscall statistics to detect performance problems. Shen uses change profiles to incrementally build predictive models of system performance, which are used to detect performance anomalies against reference workloads. OSprof

5. `strace` is included to compare `syscap` overheads against a standard tool used for manual syscall debugging.

Fault	Syscall	read-heavy workloads						Syscall	write-heavy workloads					
		ddr		iozoner		postmark			ddw		iozonew		postmark	
		TP	FP	TP	FP	TP	FP		TP	FP	TP	FP	TP	FP
disk-hog	dread	1	0	1	0	1	0.1	dwrite	0.4	0	0.4	0	1	0.1
disk-busy	dread	1	0	1	0	0.9	0.1	dwrite	0.4	0	0.4	0	1	0.1
write-network-hog								nsread	0	0	0	0	0.9	0
read-network-hog	ncread	0	0	0	0	1	0	nsread	0	0	0	0	1	0
								dwrite	0.2	0	0	0	1	0
receive-pktloss	ncread	0	0	0	0	0.8	0	nsread	0	0	0	0	0.9	0
send-pktloss	ncread	0	0	0	0	1	0							

Table 1. Results of our statistical syscall-based diagnosis. TP (FP) = true (false) positive ratio.

captures and analyzes distributions of syscall latencies under different system conditions to identify code paths that exhibit latency peaks. Our statistical analysis is similar to—but less general than—both in that it leverages peer comparison of syscall latencies across a set of nodes to establish, on the fly, a reference against which individual nodes may be compared to diagnose performance anomalies.

Ballista [16] tests code robustness by issuing function calls with exceptional arguments to elicit hangs or crashes. Although our tool is not intended for testing, we do diagnose server hangs, crashes, and even Ballista's "hindering" failures (propagated errors) as result of syscalls issued in exceptional environments.

10. Conclusion

Exploiting syscall instrumentation, we were able to detect and diagnose, with a low false-positive rate, a number of performance problems, propagated errors, and crash/hang faults in PVFS. We have effectively shown the value of syscall-based statistical and semantic analyses for diagnosing common PVFS problems. To minimize instrumentation overheads, we plan to develop a low-overhead, in-kernel utility that would allow us to efficiently use syscall tracing for continuous monitoring and online diagnosis.

Acknowledgements

We acknowledge Rob Ross, Sam Lang, Phil Carns and Kevin Harms of Argonne National Laboratory for their insightful discussions on PVFS, instrumentation sources, troubleshooting procedures, and anecdotes of performance problems in production PVFS deployments. We also thank Garth Gibson of Carnegie Mellon for feedback on this work. This research was sponsored in part by NSF grants #CCF-0621508 and by ARO agreement DAAD19–02–1–0389.

References

[1] P. H. Carns, W. B. Ligon, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for Linux clusters," in *Annual Linux Showcase and Conference*, Atlanta, GA, Oct. 2000, pp. 317–327.

[2] P. H. Carns, S. J. Lang, K. N. Harms, and R. Ross, Private communication, Dec. 2008.

[3] D. Habas and J. Sieber, "Background Patrol Read for Dell PowerEdge RAID Controllers," *Dell Power Solutions*, Feb. 2006.

[4] J. Vasileff, "latest PERC firmware == slow," Jul. 2005, http://lists.us.dell.com/pipermail/linux-poweredge/2005-July/021908.html.

[5] P. H. Carns, "need better error handling of "too many open files" condition," Jun. 2008, http://trac.mcs.anl.gov/projects/pvfs/ticket/50.

[6] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "IRON file systems," in *SOSP*, Brighton, UK, Oct. 2005, pp. 206–220.

[7] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," in *SOSP*, Brighton, UK, Oct. 2005, pp. 105–118.

[8] E. Kıcıman and A. Fox, "Detecting application-level failures in component-based Internet services," *IEEE Trans. on Neural Networks*, vol. 16, no. 5, Sep. 2005.

[9] A. V. Mirgorodskiy, "Automated problem diagnosis in distributed systems," Ph.D. dissertation, University of Wisconsin-Madison, Madison, WI, 2006.

[10] E. Thereska, B. Salmon, J. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez, and G. R. Ganger, "Stardust: Tracking activity in a distributed storage system," in *SIGMETRICS*, Saint-Malo, France, Jun. 2006.

[11] A. Aranya, C. P. Wright, and E. Zadok, "Tracefs: A file system to trace them all," in *FAST*, San Francisco, CA, Apr. 2004, pp. 129–145.

[12] S. S. Shende and A. D. Malony, "The Tau parallel performance system," *Int. J. of High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 287–311, May 2006.

[13] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, "The Paradyn parallel performance measurement tool," *IEEE Computer*, vol. 28, no. 11, pp. 37–46, Nov. 2005.

[14] K. Shen, C. Stewart, C. Li, and X. Li, "Reference-driven performance anomaly identification," in *SIGMETRICS*, Seattle, WA, Jun. 2009.

[15] N. Joukov, A. Traeger, R. Iyer, C. P. Wright, and E. Zadok, "Operating system profiling via latency analysis," in *OSDI*, Seattle, WA, Nov. 2006, pp. 89–102.

[16] J. DeVale, P. Koopman, and D. Guttendorf, "The Ballista software robustness testing service," in *Testing Computer Software*, Washington, DC, Jun. 1999.