Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST '10),
San Jose, CA, February 2010.

Black-Box Problem Diagnosis in Parallel File Systems

Michael P. Kasick[1], Jiaqi Tan[2], Rajeev Gandhi[1], Priya Narasimhan[1]

[1] *Electrical & Computer Engineering Department*
Carnegie Mellon University
Pittsburgh, PA 15213–3890
{mkasick, rgandhi, priyan}@andrew.cmu.edu

[2] *DSO National Labs, Singapore*
Singapore 118230
tjiaqi@dso.org.sg

Abstract

We focus on automatically diagnosing different performance problems in parallel file systems by identifying, gathering and analyzing OS-level, black-box performance metrics on every node in the cluster. Our peer-comparison diagnosis approach compares the statistical attributes of these metrics across I/O servers, to identify the faulty node. We develop a root-cause analysis procedure that further analyzes the affected metrics to pinpoint the faulty resource (storage or network), and demonstrate that this approach works commonly across stripe-based parallel file systems. We demonstrate our approach for realistic storage and network problems injected into three different file-system benchmarks (dd, IOzone, and PostMark), in both PVFS and Lustre clusters.

1 Introduction

File systems can experience performance problems that can be hard to diagnose and isolate. Performance problems can arise from different system layers, such as bugs in the application, resource exhaustion, misconfigurations of protocols, or network congestion. For instance, Google reported the variety of performance problems that occurred in the first year of a cluster's operation [10]: 40–80 machines saw 50% packet-loss, thousands of hard drives failed, connectivity was randomly lost for 30 minutes, 1000 individual machines failed, etc. Often, the most interesting and trickiest problems to diagnose are not the outright crash (fail-stop) failures, but rather those that result in a "limping-but-alive" system (i.e., the system continues to operate, but with degraded performance). Our work targets the diagnosis of such performance problems in parallel file systems used for high-performance cluster computing (HPC).

Large scientific applications consist of compute-intense behavior intermixed with periods of intense parallel I/O, and therefore depend on file systems that can support high-bandwidth concurrent writes. Parallel Virtual File System (PVFS) [6] and Lustre [23] are open-source, parallel file systems that provide such applications with high-speed data access to files. PVFS and Lustre are designed as client-server architectures, with many clients communicating with multiple I/O servers and one or more metadata servers, as shown in Figure 1.

Problem diagnosis is even more important in HPC where the effects of performance problems are magnified due to long-running, large-scale computations. Current diagnosis of PVFS problems involve the manual analysis of client/server logs that record PVFS operations through code-level print statements. Such (white-box) problem diagnosis incurs significant runtime overheads, and requires code-level instrumentation and expert knowledge.

Alternatively, we could consider applying existing problem-diagnosis techniques. Some techniques specify a service-level objective (SLO) first and then flag runtime SLO violations—however, specifying SLOs might be hard for arbitrary, long-running HPC applications. Other diagnosis techniques first learn the normal (i.e., fault-free) behavior of the system and then employ statistical/machine-learning algorithms to detect runtime deviations from this learned normal profile—however, it might be difficult to collect fault-free training data for all of the possible workloads in an HPC system.

We opt for an approach that does not require the specification of an SLO or the need to collect training data for all workloads. We automatically diagnose performance problems in parallel file systems by analyzing the relevant black-box performance metrics on every node. Central to our approach is our hypothesis (borne out by observations of PVFS's and Lustre's behavior) that *fault-free I/O servers exhibit symmetric (similar) trends in their storage and network metrics, while a faulty server appears asymmetric (different) in comparison*. A similar hypothesis follows for the metadata servers. From these hypotheses, we develop a statistical peer-comparison approach that automatically diagnoses the faulty server and identifies the root cause, in a parallel file-system cluster.

The advantages of our approach are that it (i) exhibits *low overhead* as collection of OS-level performance metrics imposes low CPU, memory, and network demands; (ii) *minimizes training data* for typical HPC workloads by distinguishing between workload changes and performance problems with peer-comparison; and (iii) *avoids SLOs* by being agnostic to absolute metric values in identifying whether/where a performance problem exists.

We validate our approach by studying realistic storage and network problems injected into three file-system benchmarks (dd, IOzone, and PostMark) in two parallel file systems, PVFS and Lustre. Interestingly, but perhaps unsurprisingly, our peer-comparison approach identifies the faulty node even under workload changes (usually a source of false positives for most black-box problem-diagnosis techniques). We also discuss our experiences, particularly the utility of specific metrics for diagnosis.

2 Problem Statement

Our research is motivated by the following questions: (i) *can we diagnose the faulty server in the face of a performance problem in a parallel file system*, and (ii) *if so, can we determine which resource (storage or network) is causing the problem?*

Goals. Our approach should exhibit:

- *Application-transparency* so that PVFS/Lustre applications do not require any modification. The approach should be independent of PVFS/Lustre operation.
- *Minimal false alarms* of anomalies in the face of legitimate behavioral changes (e.g., workload changes due to increased request rate).
- *Minimal instrumentation overhead* so that instrumentation and analysis does not adversely impact PVFS/Lustre's operation.
- *Specific problem coverage* that is motivated by anecdotes of performance problems in a production parallel file-system deployment (see § 4).

Non-Goals. Our approach does not support:

- *Code-level debugging.* Our approach aims for coarse-grained problem diagnosis by identifying the culprit server, and where possible, the resource at fault. We currently do not aim for fine-grained diagnosis that would trace the problem to lines of PVFS/Lustre code.
- *Pathological workloads.* Our approach relies on I/O servers exhibiting similar request patterns. In parallel file systems, the request pattern for most workloads is similar across all servers—requests are either large enough to be striped across all servers or random enough to result in roughly uniform access. However, some workloads (e.g., overwriting the same portion of a file repeatedly, or only writing stripe-unit-sized records to every stripe-count offset) make requests distributed to only a subset, possibly one, of the servers.
- *Diagnosis of non-peers.* Our approach fundamentally cannot diagnose performance problems on non-peer nodes (e.g., Lustre's single metadata server).

Hypotheses. We hypothesize that, under a performance fault in a PVFS or Lustre cluster, OS-level performance metrics should exhibit observable anomalous behavior on the culprit servers. Additionally, with knowledge of PVFS/Lustre's overall operation, we hypothesize that the statistical trends of these performance data: (i) should be similar (albeit with inevitable minor differences) across fault-free I/O servers, even under workload changes, and (ii) will differ on the culprit I/O server, as compared to the fault-free I/O servers.

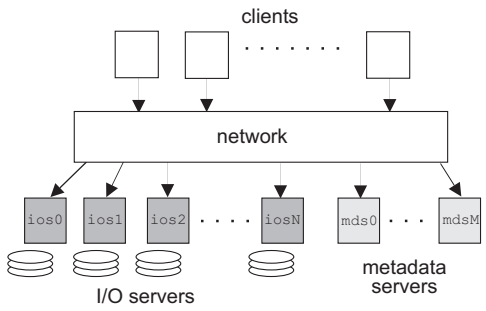

Figure 1: Architecture of parallel file systems, showing the I/O servers and the metadata servers.

Assumptions. We assume that a majority of the I/O servers exhibit fault-free behavior, that all peer server nodes have identical software configurations, and that the physical clocks on the various nodes are synchronized (e.g., via NTP) so that performance data can be temporally correlated across the system. We also assume that clients and servers are comprised of homogeneous hardware and execute homogeneous workloads. These assumptions are reasonable in HPC environments where homogeneity is both deliberate and critical to large scale operation. Homogeneity of hardware and client workloads is not strictly required for our diagnosis approach (§ 12 describes our experience with heterogeneous hardware). However we have not yet tested our approach with deliberately heterogeneous hardware or workloads.

3 Background: PVFS & Lustre

PVFS clusters consist of one or more metadata servers and multiple I/O servers that are accessed by one or more PVFS clients, as shown in Figure 1. The PVFS server consists of a single monolithic user-space daemon that may act in either or both metadata and I/O server roles.

PVFS clients consist of stand-alone applications that use the PVFS library (*libpvfs2*) or MPI applications that use the ROMIO MPI-IO library (that supports PVFS internally) to invoke file operations on one or more servers. PVFS can also plug in to the Linux Kernel's VFS interface via a kernel module that forwards the client's syscalls (requests) to a user-space PVFS client daemon that then invokes operations on the servers. This kernel client allows PVFS file systems to be mounted under Linux similar to other remote file systems like NFS.

With PVFS, file-objects are distributed across all I/O servers in a cluster. In particular, file data is striped

across each I/O server with a default stripe size of 64 kB. For each file-object, the first stripe segment is located on the I/O server to which the object handle is assigned. Subsequent segments are accessed in a round-robin manner on each of the remaining I/O servers. This characteristic has significant implications on PVFS's throughput in the event of a performance problem.

Lustre clusters consist of one active metadata server which serves one metadata target (storage space), one management server which may be colocated with the metadata server, and multiple object storage servers which serve one or more object storage targets each. The metadata and object storage servers are analogous to PVFS's metadata and I/O servers with the main distinction of only allowing for a single active metadata server per cluster. Unlike PVFS, the Lustre server is implemented entirely in kernel space as a loadable kernel module. The Lustre client is also implemented as a kernel space file-system module, and like PVFS, provides file system access via the Linux VFS interface. A userspace client library (*liblustre*) is also available.

Lustre allows for the configurable striping of file data across one or more object storage targets. By default, file data is stored on a single target. The `stripe_count` parameter may be set on a per-file, directory, or file-system basis to specify the number of object storage targets that file data is striped over. The `stripe_size` parameter specifies the stripe unit size and may be configured to multiples of 64 kB, with a default of 1 MB (the maximum payload size of a Lustre RPC).

4 Motivation: Real Problem Anecdotes

The faults we study here are motivated by the PVFS developers' anecdotal experience [5] 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. Accounts of experience with BG/P indicate that storage/network problems account for approximately 50%/50% of performance issues [5]. A single poorly performing server has been observed to impact the behavior of the overall system, instead of its behavior being averaged out by that of non-faulty nodes [5]. This makes it difficult to troubleshoot system-wide performance issues, and thus, fault localization (i.e., diagnosing the faulty server) is a critical first step in root-cause analysis.

Anomalous storage behavior can result from a number of causes. Aside from failing disks, RAID controllers may scan disks during idle times to proactively search for media defects [13], inadvertently creating disk contention that degrades the throughput of a disk array [25]. Our *disk-busy* injected problem (§ 5) seeks to emulate this manifestation. Another possible cause of a disk-busy problem is disk contention due to the accidental launch of a rogue processes. For example, if two remote file servers (e.g., PVFS and GPFS) are collocated, the startup of a second server (GPFS) might negatively impact the performance of the server already running (PVFS) [5].

Network problems primarily manifest in packet-loss errors, which is reported to be the "most frustrating" [sic] to diagnose [5]. Packet loss is often the result of faulty switch ports that enter a degraded state when packets can still be sent but occasionally fail CRC checks. The resulting poor performance spreads through the rest of the network, making problem diagnosis difficult [5]. Packet loss might also be the result of an overloaded switch that "just can't keep up" [sic]. In this case, network diagnostic tests of individual links might exhibit no errors, and problems manifest only while PVFS is running [5].

Errors do not necessarily manifest identically under all workloads. For example, SANs with large write caches can initially mask performance problems under write-intensive workloads and thus, the problems might take a while to manifest [5]. In contrast, performance problems in read-intensive workloads manifest rather quickly.

A consistent, but unfortunate, aspect of performance faults is that they result in a "limping-but-alive" mode, where system throughput is drastically reduced, but the system continues to run without errors being reported. Under such conditions, it is likely not possible to identify the faulty node by examining PVFS/application logs (neither of which will indicate any errors) [5].

Fail-stop performance problems usually result in an outright server crash, making it relatively easy to identify the faulty server. Our work targets the diagnosis of non-fail-stop performance problems that can degrade server performance without escalating into a server crash. There are basically three resources—CPU, storage, network—being contended for that are likely to cause throughput degradation. CPU is an unlikely bottleneck as parallel file systems are mostly I/O-intensive, and fair CPU scheduling policies should guarantee that enough time-slices are available. Thus, we focus on the remaining two resources, storage and network, that are likely to pose performance bottlenecks.

5 Problems Studied for Diagnosis

We separate problems involving storage and network resources into two classes. The first class is *hog* faults, where a rogue process on the monitored file servers induces an unusually high workload for the specific resource. The second class is *busy* or *loss* faults, where an unmonitored (i.e., outside the scope of the server OSes) third party creates a condition that causes a performance degradation for the specific resource. To explore all combinations of problem resource and class, we study the diagnosis of four problems—disk-hog, disk-busy, network-hog, packet-loss (network-busy).

Metric [s/n]*	Significance
`tps` [s]	Number of I/O (read and write) requests made to the disk per second.
`rd_sec` [s]	Number of sectors read from disk per second.
`wr_sec` [s]	Number of sectors written to disk per second.
`avgrq-sz` [s]	Average size (in sectors) of disk I/O requests.
`avgqu-sz` [s]	Average number of queued disk I/O requests; generally a low integer (0–2) when the disk is under-utilized; increases to ≈100 as disk utilization saturates.
`await` [s]	Average time (in milliseconds) that a request waits to complete; includes queuing delay and service time.
`svctm` [s]	Average service time (in milliseconds) of I/O requests; is the pure disk-servicing time; does not include any queuing delay.
`%util` [s]	Percentage of CPU time in which I/O requests are made to the disk.
`rxpck` [n]	Packets received per second.
`txpck` [n]	Packets transmitted per second.
`rxbyt` [n]	Bytes received per second.
`txbyt` [n]	Bytes transmitted per second.
`cwnd` [n]	Number of segments (per socket) allowed to be sent outstanding without acknowledgment.

*Denotes storage (s) or network (n) related metric.

Table 1: Black-box, OS-level performance metrics collected for analysis.

Disk-hogs can result from a runaway, but otherwise benign, process. They may occur due to unexpected `cron` jobs, e.g., an `updatedb` process generating a file/directory index for GNU `locate`, or a monthly software-RAID array verification check. Disk-busy faults can also occur in shared-storage systems due to a third-party/unmonitored node that runs a disk-hog process on the shared-storage device; we view this differently from a regular disk-hog because the increased load on the shared-storage device is not observable as a throughput increase at the monitored servers.

Network-hogs can result from a local traffic-emitter (e.g., a backup process), or the receipt of data during a denial-of-service attack. Network-hogs are observable as increased throughput (but not necessarily "goodput") at the monitored file servers. Packet-loss faults might be the result of network congestion, e.g., due to a network-hog on a nearby unmonitored node or due to packet corruption and losses from a failing NIC.

6 Instrumentation

For our problem diagnosis, we gather and analyze OS-level performance metrics, without requiring any modifications to the file system, the applications or the OS.

In Linux, OS-level performance metrics are made available as text files in the `/proc` pseudo file system. Table 1 describes the specific metrics that we collect. Most `/proc` data is collected via sysstat 7.0.0's `sadc` program [12]. `sadc` is used to periodically gather storage- and network-related metrics (as we are primarily concerned with performance problems due to storage and network resources, although other kinds of metrics are available) at a sampling interval of one second. For storage resources sysstat provides us with throughput (`tps`, `rd_sec`, `wr_sec`) and latency (`await`, `svctm`) metrics, and for network resources it provides us with throughput (`rxpck`, `txpck`, `rxbyt`, `txbyt`) metrics.

Unfortunately sysstat provides us only with throughput data for network resources. To obtain congestion data as well, we sample the contents of `/proc/net/tcp`, on both clients and servers, once every second. This gives us TCP congestion-control data [22] in the form of the sending congestion-window (`cwnd`) metric.

6.1 Parallel File-System Behavior

We highlight our (empirical) observations of PVFS's/Lustre's behavior that we believe is characteristic of stripe-based parallel file systems. Our preliminary studies of two other parallel file systems, GlusterFS [2] and Ceph [26], also reveal similar insights, indicating that our approach might apply to parallel file systems in general.

[Observation 1] *In a homogeneous (i.e., identical hardware) cluster, I/O servers track each other closely in throughput and latency, under fault-free conditions.*

For N I/O servers, I/O requests of size greater than $(N-1) \times stripe\_size$ results in I/O on each server for a single request. Multiple I/O requests on the same file, even for smaller request sizes, will quickly generate workloads[1] on all servers. Even I/O requests to files smaller than *stripe_size* will generate workloads on all I/O servers, as long as enough small files are read/written. We observed this for all three target benchmarks, dd, IOzone, and PostMark. For metadata-intensive workloads, we expect that metadata servers also track each other in proportional magnitudes of throughput and latency.

[Observation 2] *When a fault occurs on at least one of the I/O servers, the other (fault-free) I/O servers experience an identical drop in throughput.*

When a client syscall involves requests to multiple I/O servers, the client must wait for all of these servers to respond before proceeding to the next syscall.[2] Thus, the client-perceived cluster performance is constrained by the slowest server. We call this the *bottlenecking condition*. When a server experiences a performance fault, that server's per-request service-time increases. Because the

[1] Pathological workloads might not result in equitable workload distribution across I/O servers; one server would be disproportionately deluged with requests, while the other servers are idle, e.g., a workload that constantly rewrites the same *stripe_size* chunk of a file.

[2] Since Lustre performs client side caching and readahead, client I/O syscalls may return immediately even if the corresponding file server is faulty. Even so, a maximum of 32 MB may be cached (or 40 MB pre-read) before Lustre must wait for responses.

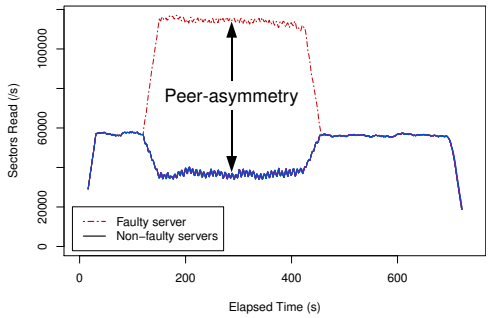

Figure 2: Peer-asymmetry of `rd_sec` for `iozoner` workload with *disk-hog* fault.

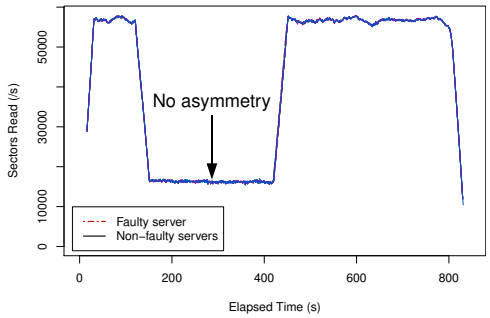

Figure 3: No asymmetry of `rd_sec` for `iozoner` workload with *disk-busy* fault.

client blocks on the syscall until it receives all server responses, the client's syscall-service time also increases. This leads to slower application progress and fewer requests per second from the client, resulting in a proportional decrease in throughput on all I/O servers.

[Observation 3] *When a performance fault occurs on at least one of the I/O servers, the other (fault-free) I/O servers are unaffected in their per-request service times.*

Because there is no server-server communication (i.e., no server inter-dependencies), a performance problem at one server will not adversely impact latency (per-request service-time) at the other servers. If these servers were previously highly loaded, latency might even improve (due to potentially decreased resource contention).

[Observation 4] *For disk/network-hog faults, storage/network-throughput increases at the faulty server and decreases at the non-faulty servers.*

A disk/network-hog fault at a server is due to a third-party that creates additional I/O traffic that is observed as increased storage/network-throughput. The additional I/O traffic creates resource contention that ultimately manifests as a decrease in file-server throughput on all servers (causing the bottlenecking condition of observation 2). Thus, disk- and network-hog faults can be localized to the faulty server by looking for *peer-divergence* (i.e. *asymmetry* across peers) in the storage- and network-throughput metrics, respectively, as seen in Figure 2.

[Observation 5] *For disk-busy (packet-loss) faults, storage- (network-) throughput decreases on all servers.*

For disk-busy (packet-loss) faults, there is no asymmetry in storage (network) throughputs across I/O servers (because there is no other process to create observable throughput, and the server daemon has the same throughput at all the nodes). Instead, there is a symmetric decrease in the storage-(network-) throughput metrics across all servers. Because asymmetry does not arise, such faults cannot be diagnosed, as seen in Figure 3.

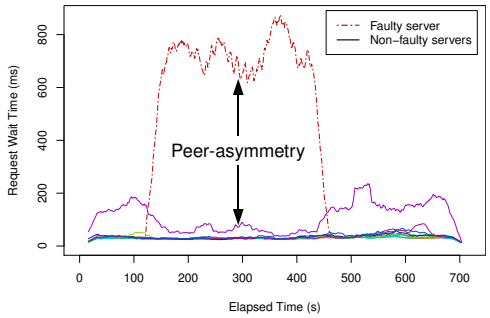

Figure 4: Peer-asymmetry of `await` for `ddr` workload with *disk-hog* fault.

[Observation 6] *For disk-busy and disk-hog faults, storage-latency increases on the faulty server and decreases at the non-faulty servers.*

For disk-busy and disk-hog faults, `await`, `avgqu-sz` and `%util` increase at the faulty server as the disk's responsiveness decreases and requests start to backlog. The increased `await` on the faulty server causes an increased server response-time, making the client wait longer before it can issue its next request. The additional delay that the client experiences reduces its I/O throughput, resulting in the fault-free servers having increased idle time. Thus, the `await` and `%util` metrics decrease asymmetrically on the fault-free I/O servers, enabling a peer-comparison diagnosis of the disk-hog and disk-busy faults, as seen in Figure 4.

[Observation 7] *For network-hog and packet-loss faults, the TCP congestion-control window decreases significantly and asymmetrically on the faulty server.*

The goal of TCP congestion control is to allow `cwnd` to be as large as possible, without experiencing packet-loss due to overfilling packet queues. When packet-loss occurs and is recovered within the retransmission timeout interval, the congestion window is halved. If recovery takes longer than retransmission timeout, `cwnd` is reduced to one segment. When nodes are transmitting data, their `cwnd` metrics either stabilize at high (≈100) val-

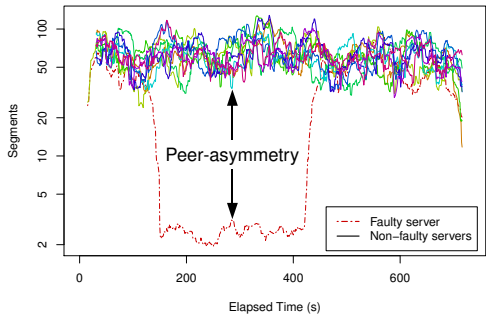

Figure 5: Peer-asymmetry of `cwnd` for `ddw` workload with *receive-pktloss* fault.

ues or oscillate (between ≈10–100) as congestion is observed on the network. However, during (some) network-hog and (all) packet-loss experiments, `cwnd`s of connections to the faulty server dropped by several orders of magnitude to single-digit values and held steady until the fault was removed, at which time the congestion window was allowed to open again. These asymmetric sustained drops in `cwnd` enable peer-comparison diagnosis for network faults, as seen in Figure 5.

7 Discussion on Metrics

Although faults present in multiple metrics, not all metrics are appropriate for diagnosis as they exhibit inconsistent behaviors. Here we describe problematic metrics.

Storage-throughput metrics. There is a notable relationship between the storage-throughput metrics: `tps` × `avgrq-sz` = `rd_sec` + `wr_sec`. While `rd_sec` and `wr_sec` accurately capture real storage activity and strongly correlate across I/O servers, `tps` and `avgrq-sz` do not correlate as strongly because a lower transfer rate may be compensated by issuing larger-sized requests. Thus, `tps` is not a reliable metric for diagnosis.

`svctm`. The impact of disk faults on `svctm` is inconsistent. The influences on storage service times are: time to locate the starting sector (seek time and rotational delay), media-transfer time, reread/rewrite time in the event of a read/write error, and delay time to due servicing of unobservable requests. During a disk fault, servicing of interleaved requests increases seek time. Thus, for an unchanged `avgrq-sz`, `svctm` will increase asymmetrically on the faulty server. Furthermore, during a disk-busy fault, servicing of unobservable requests further increases `svctm` due to request delays. However, during a disk-hog fault, the hog process might be issuing requests of smaller sizes than PVFS/Lustre. If so, then the associated decrease in media-transfer time might offset the increase in seek time resulting in a decreased or unchanged `svctm`. Thus, `svctm` is not guaranteed to exhibit asymmetries for disk-hogs, and therefore is unreliable.

Other metrics. While problems manifest on other metrics (e.g., CPU usage, context-switch rate), these secondary manifestations are due to the overall reduction in I/O throughput during the faulty period, and reveal nothing new. Thus, we do not analyze these metrics.

8 Experimental Set-Up

We perform our experiments on AMD Opteron 1220 machines, each with 4 GB RAM, two Seagate Barracuda 7200.10 320 GB disks (one dedicated for PVFS/Lustre storage), and a Broadcom NetXtreme BCM5721 Gigabit Ethernet controller. Each node runs Debian GNU/Linux 4.0 (etch) with Linux kernel 2.6.18. The machines run in stock configuration with background tasks turned off. We conduct experiments with x/y configurations, i.e., the PVFS x/y cluster comprises y combined I/O and metadata servers and x clients, while the equivalent Lustre x/y cluster comprises y object storage (I/O) servers with a single object storage target each, a single (dedicated) metadata server, and x clients. We conduct our experiments for 10/10 and 6/12 PVFS and Lustre clusters;[3] in the interests of space, we explain the 10/10 cluster experiments in detail, but our observations carry to both.

For these experiments PVFS 2.8.0 is used in the default server (`pvfs2-genconfig` generated) configuration with two modifications. First, we use the Direct I/O method (`TroveMethod directio`) to bypass the Linux buffer cache for PVFS I/O server storage. This is required for diagnosis as we otherwise observe disparate I/O server behavior during IOzone's rewrite phase. Although bypassing the buffer cache has no effect on diagnosis for non-rewrite (e.g., `ddw`) workloads, it does improve large write throughput by 10%.

Second, we increase to 4 MB (from 256 kB) the Flow buffer size (`FlowBufferSizeBytes`) to allow larger bulk data transfers and enable more efficient disk usage. This modification is standard practice in PVFS performance tuning, and is required to make our testbed performance representative of real deployments. It does not appear to affect diagnosis capability. In addition, we patch the PVFS kernel client to eliminate the 128 MB total size restriction on the `/dev/pvfs2-req` device request buffers and to `vmalloc` memory (instead of `kmalloc`) for the buffer page map (`bufmap_page_array`) to ensure that larger request buffers are actually allocatable. We then invoke the PVFS kernel client with 64 MB request buffers (`desc-size` parameter) in order to make the 4 MB data transfers to each of the I/O servers.

For Lustre experiments we use the etch backport of the Lustre 1.6.6 Debian packages in the default

[3] Due to a limited number of nodes we were unable to experiment with higher active client/server ratios. However, with the workloads and faults tested, an increased number of clients appears to degrade per-client throughput with no significant change in other behavior.

server configuration with a single modification to set the `lov.stripecount` parameter to −1 to stripe files across each object storage target (I/O server).

The nodes are rebooted immediately prior to the start of each experiment. Time synchronization is performed at boot-time using `ntpdate`. Once the servers are initialized and the client is mounted, monitoring agents start capturing metrics to a local (non-storage dedicated) disk. `sync` is then performed, followed by a 15-second sleep, and the experiment benchmark is run. The benchmark runs fault-free for 120 seconds prior to fault injection. The fault is then injected for 300 seconds and then deactivated. The experiment continues to the completion of the benchmark, which ideally runs for a total of 600 seconds in the fault-free case. This run time allows the benchmark to run for at least 180 seconds after a fault's deactivation to determine if there are any delayed effects. We run ten experiments for each workload & fault combination, using a different faulty server for each iteration.

8.1 Workloads

We use five experiment workloads derived from three experiment benchmarks: dd, IOzone, and PostMark. The same workload is invoked concurrently on all clients. The first two workloads, `ddw` and `ddr`, either write zeros (from `/dev/zero`) to a client-specific temporary file or read the contents of a previously written client-specific temporary file and write the output to `/dev/null`.

`dd` [24] performs a constant-rate, constant-workload large-file read/write from/to disk. It is the simplest large-file benchmark to run, and helps us to analyze and understand the system's behavior prior to running more complicated workloads. `dd` models the behavior of scientific-computing workloads with constant data-write rates.

Our next two workloads, `iozonew` and `iozoner`, consist of the same file-system benchmark, IOzone v3.283 [4]. We run `iozonew` in write/rewrite mode and `iozoner` in read/reread mode. IOzone's behavior is similar to `dd` in that it has two constant read/write phases. Thus, IOzone is a large-file I/O-heavy benchmark with few metadata operations. However, there is an `fsync` and a workload change half-way through.

Our fifth benchmark is PostMark v1.51 [15]. PostMark was chosen as a metadata-server heavy workload with small file writes (all writes < 64 kB thus, writes occur only on a single I/O server per file).

Configurations of Workloads. For the `ddw` workload, we use a 17 GB file with a record-size of 40 MB for PVFS, and a 30 GB file is used with a record-size 10 MB for Lustre. File sizes are chosen to result in a fault-free experiment runtime of approximately 600 seconds. The PVFS record-size was chosen to result in 4 MB bulk data transfers to each I/O server, which we empirically determined to be the knee of the performance vs. record-size curve. The Lustre record-size was chosen to result in 1 MB bulk data transfers to each I/O server—the maximum payload size of a Lustre RPC. Since Lustre both aggregates client writes and performs readahead, varying the record-size does not significantly alter Lustre read or write performance. For `ddr` we use a 27 GB file with a record-size of 40 MB for PVFS, and a 30 GB file with a record-size of 10 MB for Lustre (same as `ddw`).

For both the `iozonew` and `iozoner` workloads, we use an 8 GB file with a record-size of 16 MB (the largest that IOzone supports) for PVFS. For Lustre we use a 9 GB file with a record-size of 10 MB for `iozonew`, and a 16 GB file with the same record-size for `iozoner`. For `postmark` we use its default configuration with 16,000 transactions for PVFS and 53,000 transactions for Lustre to give a sufficiently long-running benchmark.

9 Fault Injection

In our fault-induced experiments, we inject a single fault at a time into one of the I/O servers to induce degraded performance for either network or storage resources. We inject the following faults:

- *disk-hog*: a `dd` process that reads 256 MB blocks (using direct I/O) from an unused storage disk partition.
- *disk-busy*: an `sgm_dd` process [11] that issues low-level SCSI I/O commands via the Linux SCSI Generic (`sg`) driver to read 1 MB blocks from the same unused storage disk partition.
- *network-hog*: a third-party node opens a TCP connection to a listening port on one of the PVFS I/O servers and sends zeros to it (*write-network-hog*), or an I/O server opens a connection and sends zeros to a third party node (*read-network-hog*).
- *pktloss*: a netfilter firewall rule that (probabilistically) drops packets received at one of the I/O servers with probability 5% (*receive-pktloss*), or a firewall rule on all clients that drops packets incoming from a single server with probability 5% (*send-pktloss*).

10 Diagnosis Algorithm

The first phase of the peer-comparison diagnostic algorithm identifies the faulty I/O server for the faults studied. The second phase performs root-cause analysis to identify the resource at fault.

10.1 Phase I: Finding the Faulty Server

We considered several statistical properties (e.g., the mean, the variance, etc. of a metric) as candidates for peer-comparison across servers, but ultimately chose the probability distribution function (PDF) of each metric because it captures many of the metric's statistical properties. Figure 6 shows the asymmetry in a metric's histograms/PDFs between the faulty and fault-free servers.

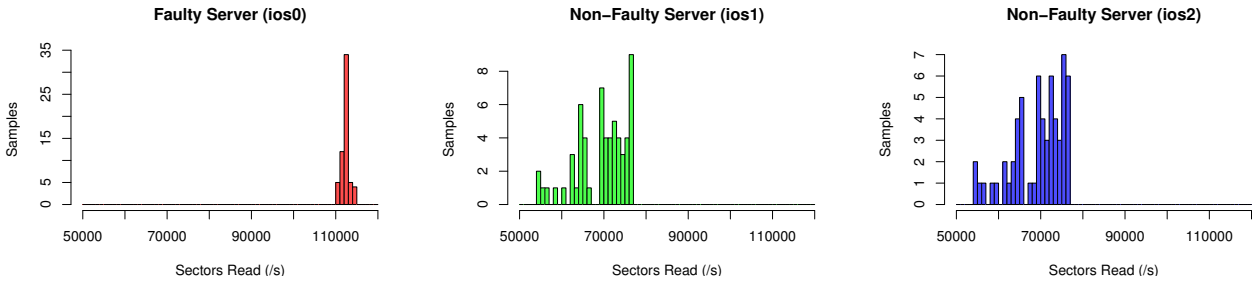

Figure 6: Histograms of `rd_sec` (`ddr` with *disk-hog* fault) for one faulty and two non-faulty servers.

Histogram-Based Approach. We determine the PDFs, using histograms as an approximation, of a specific black-box metric values over a window of time (of size *WinSize* seconds) at each I/O server. To compare the resulting PDFs across the different I/O servers, we use a standard measure, the Kullback-Leibler (KL) divergence [9], as the distance between two distribution functions, P and Q.[4] The KL divergence of a distribution function, Q, from the distribution function, P, is given by $D(P||Q) = \sum_i P(i) \log \frac{P(i)}{Qi}$. We use a symmetric version of the KL divergence, given by $D'(P||Q) = \frac{1}{2}[D(P||Q) + D(Q||P)]$ in our analysis.

We perform the following procedure for each of metric of interest. Using i to represent one of these metrics, we first perform a moving average on i. We then take PDFs of the smoothed i for two distinct I/O servers at a time and compute their pairwise KL divergences. A pairwise KL-divergence value for i is flagged as anomalous if it is greater than a certain predefined threshold. An I/O server is flagged as anomalous if its pairwise KL-divergence for i is anomalous with more than half of the other servers for at least k of the past $2k-1$ windows. The window is shifted in time by $WinShift$ (there is an overlap of $WinSize - WinShift$ samples between two consecutive windows), and the analysis is repeated. A server is indicted as faulty if it is anomalous in one or more metrics.

We use a 5-point moving average to ensure that metrics reflect average behavior of request processing. We also use a *WinSize* of 64, a $WinShift$ of 32, and a k of 3 in our analysis to incorporate a reasonable quantity of data samples per comparison while maintaining a reasonable diagnosis latency (approximately 90 seconds). We investigate the useful ranges of these values in § 11.2.

Time Series-Based Approach. We use the histogram-based approach for all metrics except `cwnd`. Unlike other metrics, `cwnd` tends to be noisy under normal conditions. This is expected as TCP congestion control prevents synchronized connections from fully utilizing link capacity. Thus `cwnd` analysis is different from other metrics as there is no closely-coupled peer behavior.

[4] Alternatively, earth mover's distance [20] or another distance measure may be used instead of KL.

Fortunately, there is a simple heuristic for detecting packet-loss using `cwnd`. TCP congestion control responds to packet-loss by halving `cwnd`, which results `cwnd` exponential decay after multiple loss events. When viewed on a logarithmic scale, sustained packet-loss results in a linear decrease for each packet lost.

To support analysis of `cwnd`, we first generate a time-series by performing a moving average on `cwnd` with a window size of 31 seconds. Based on empirical observation, this attenuates the effect of sporadic transmission timeout events while enabling reasonable diagnosis latencies (i.e., under one minute). Then, every second, a representative value (median) is computed of the log-`cwnd` values. A server is indicted if its log-`cwnd` is less than a predetermined fraction (threshold) of the median.

Threshold Selection. Both the histogram and time-series analysis algorithms require thresholds to differentiate between faulty and fault-free servers. We determine the thresholds through a fault-free training phase that captures a profile of relative server performance.

We do not need to train against all potential workloads, instead we train on workloads that are expected to stress the system to its limits of performance. Since server performance deviates the most when resources are saturated (and thus, are unable to "keep up" with other nodes), these thresholds represent the maximum expected performance deviations under normal operation. Less intense workloads, since they do not saturate server resources, are expected to exhibit better coupled peer behavior.

As the training phase requires training on the specific file system and hardware intended for problem diagnosis, we recommend training with HPC workloads normally used to stress-test systems for evaluation and purchase. Ideally these tests exhibit worst-case request rates, payload sizes, and access patterns expected during normal operation so as to saturate resources, and exhibit maximally-expected request queuing. In our experiments, we train with 10 iterations of the `ddr`, `ddw`, and `postmark` fault-free workloads. The same metrics are captured during training as when performing diagnosis.

To train the histogram algorithm, for each metric, we start with a minimum threshold value (currently 0.1) and

increase in increments (of 0.1) until the minimum threshold is determined that eliminates all anomalies on a particular server. This server-specific threshold is doubled to provide a cushion that masks minor manifestations occurring during the fault period. This is based on the premise that a fault's primary manifestation will cause a metric to be sufficiently asymmetric, roughly an order of magnitude, yielding a "safe window" of thresholds that can be used without altering the diagnosis.

Training the time-series algorithm is similar, except that the final threshold is not doubled as the `cwnd` metric is very sensitive, yielding a much smaller corresponding "safe window". Also, only two thresholds are determined for `cwnd`, one for all servers sending to clients, and one for clients sending to servers. As `cwnd` is generally not influenced by the performance of specific hardware, its behavior is consistent across nodes.

10.2 Phase II: Root-Cause Analysis

In addition to identifying the faulty server, we also infer the resource that is the root cause of the problem through an expert derived checklist. This checklist, based on our observations (§ 6.1) of PVFS's/Lustre's behavior, maps sets of peer-divergent metrics to the root cause. Where multiple metrics may be used, the specific metrics selected are chosen for consistency of behavior (see § 7). If we observe peer-divergence at any step of the checklist, we halt at that step and arrive at the root cause and faulty server. If peer-divergence is not observed at that step, we continue to the next step of decision-making.

Do we observe peer-divergence in ...

1. Storage throughput? (`rd_sec` or `wr_sec`) — Yes: disk-hog fault; No: next question
2. Storage latency? (`await`) — Yes: disk-busy fault; No: ...
3. Network throughput?* (`rxbyt` or `txbyt`) — Yes: network-hog fault; No: ...
4. Network congestion? (`cwnd`) — Yes: packet-loss fault; No: no fault discovered

*Must diverge in both `rxbyt` & `txbyt`, or in absence of peer-divergence in `cwnd` (see § 12).

11 Results

PVFS Results. Tables 2 and 3 shows the accuracy (true- and false-positive rates) of our diagnosis algorithm in indicting faulty nodes (ITP/IFP) and diagnosing root causes (DTP/DFP)[5] for the PVFS 10/10 & 6/12 clusters.

[5]ITP is the percentage of experiments where all faulty servers are correctly indicted as faulty, IFP is the percentage where at least one non-faulty server is misindicted as faulty. DTP is the percentage of experiments where all faults are successfully diagnosed to their root causes, DFP is the percentage where at least one fault is misdiagnosed to a wrong root cause (including misindictments).

Fault	ITP	IFP	DTP	DFP
None (control)	0.0%	0.0%	0.0%	0.0%
disk-hog	100.0%	0.0%	100.0%	0.0%
disk-busy	90.0%	2.0%	90.0%	2.0%
write-network-hog	92.0%	0.0%	84.0%	8.0%
read-network-hog	100.0%	0.0%	100.0%	0.0%
receive-pktloss	42.0%	0.0%	42.0%	0.0%
send-pktloss	40.0%	0.0%	40.0%	0.0%
Aggregate	77.3%	0.3%	76.0%	1.4%

Table 2: Results of PVFS diagnosis for the 10/10 cluster.

Fault	ITP	IFP	DTP	DFP
None (control)	0.0%	2.0%	0.0%	2.0%
disk-hog	100.0%	0.0%	100.0%	0.0%
disk-busy	100.0%	0.0%	100.0%	0.0%
write-network-hog	42.0%	2.0%	0.0%	44.0%
read-network-hog	0.0%	2.0%	0.0%	2.0%
receive-pktloss	54.0%	6.0%	54.0%	6.0%
send-pktloss	40.0%	2.0%	40.0%	2.0%
Aggregate	56.0%	2.0%	49.0%	8.0%

Table 3: Results of PVFS diagnosis for the 6/12 cluster.

It is notable that not all faults manifest equally on all workloads. *disk-hog*, *disk-busy*, and *read-network-hog* all exhibit a significant (> 10%) runtime increase for all workloads. In contrast, the *receive-pktloss* and *send-pktloss* only have significant impact on runtime for write-heavy and read-heavy workloads respectively. Correspondingly, faults with greater runtime impact are often the most reliably diagnosed. Since packet-loss faults have negligible impact on `ddr` & `ddw` ACK flows and `postmark` (where lost packets are recovered quickly), it is reasonable to expect to not be able to diagnose them.

When removing the workloads for which packet-loss cannot be observed (and thus, not diagnosed), the aggregate diagnosis rates improve to 96.3% ITP and 94.6% DTP in the 10/10 cluster, and to 67.2% ITP and 58.8% DTP in the 6/12 cluster.

Lustre Results. Tables 4 and 5 shows the accuracy of our diagnosis algorithm for the Lustre 10/10 & 6/12 clusters. When removing workloads for which packet-loss cannot be observed, the aggregate diagnosis rates improve to 92.5% ITP and 86.3% DTP in the 10/10 cluster, and to 90.0% ITP and 82.1% DTP in the 6/12 case.

Both 10/10 clusters exhibit comparable accuracy rates. In contrast, the PVFS 6/12 cluster exhibits masked network-hogs faults (fewer true-positives) due to low network throughput thresholds from training with unbalanced metadata request workloads (see § 12). The Lustre 6/12 cluster exhibits more misdiagnoses (higher false-positives) due to minor, secondary manifestations in storage throughput. This suggests that our analysis algorithm may be refined with a ranking mechanism that allows diagnosis to tolerate secondary manifestations (see § 14).

Fault	ITP	IFP	DTP	DFP
None (control)	0.0%	0.0%	0.0%	0.0%
disk-hog	82.0%	0.0%	82.0%	0.0%
disk-busy	88.0%	2.0%	68.0%	22.0%
write-network-hog	98.0%	2.0%	96.0%	4.0%
read-network-hog	98.0%	2.0%	94.0%	6.0%
receive-pktloss	38.0%	4.0%	36.0%	6.0%
send-pktloss	40.0%	0.0%	38.0%	2.0%
Aggregate	74.0%	1.4%	69.0%	5.7%

Table 4: Results of Lustre diagnosis for the 10/10 cluster.

Fault	ITP	IFP	DTP	DFP
None (control)	0.0%	6.0%	0.0%	6.0%
disk-hog	100.0%	0.0%	100.0%	0.0%
disk-busy	76.0%	8.0%	38.0%	46.0%
write-network-hog	86.0%	14.0%	86.0%	14.0%
read-network-hog	92.0%	8.0%	92.0%	8.0%
receive-pktloss	40.0%	2.0%	40.0%	2.0%
send-pktloss	38.0%	8.0%	38.0%	8.0%
aggregate	72.0%	6.6%	65.7%	12.0%

Table 5: Results of Lustre diagnosis for the 6/12 cluster.

11.1 Diagnosis Overheads & Scalability

Instrumentation Overhead. Table 6 reports runtime overheads for instrumentation of both PVFS and Lustre for our five workloads. Overheads are calculated as the increase in mean workload runtime (for 10 iterations) with respect to their uninstrumented counterparts. Negative overheads are result of sampling error, which is high due runtime variance across experiments. The PVFS workload with the least runtime variance (`iozoner`) exhibits, with 99% confidence, a runtime overhead < 1%. As the server load of this workload is comparable to the others, we conclude that OS-level instrumentation has negligible impact on throughput and performance.

Data Volume. The performance metrics collected by `sadc` have an uncompressed data volume of 3.8 kB/s on each server node, independent of workload or number of clients. The congestion-control metrics sampled from `/proc/net/tcp` have a data volume of 150 B/s per socket on each client & server node. While the volume of congestion-control data linearly increases with number of clients, it is not necessary to collect per-socket data for all clients. At minimum, congestion-control data needs to be collected for only a single active client per time window. Collecting congestion-control data from additional clients merely ensures that server packet-loss effects are observed by a representative number of clients.

Algorithm Scalability. Our analysis code requires, every second, 3.44 ms per server and 182 µs per server pair of CPU time on a 2.4 GHz dedicated core to diagnose a fault if any exists. Therefore, realtime diagnosis of up to 88 servers may be supported on a single 2.4 GHz core.

Although the pairwise analysis algorithm is $O(n^2)$, we recognize that it is not necessary to compare a given server against all others in every analysis window. To support very large clusters (thousands of servers), we recommend partitioning n servers into $n-k$ analysis domains of k (e.g., 10) servers each, and only performing pairwise comparisons within these partitions. To avoid undetected anomalies that might develop in static partitions, we recommend rotating partition membership in each analysis window. Although we have not yet tested this technique, it does allow for $O(n)$ scalability.

Overhead for Workload	File System	
	PVFS	Lustre
`ddr`	0.90% ± 0.62%	1.81% ± 1.71%
`ddw`	0.00% ± 1.03%	–0.22% ± 1.18%
`iozoner`	–0.07% ± 0.37%	0.70% ± 0.98%
`iozonew`	–0.77% ± 1.62%	0.53% ± 2.71%
`postmark`	–0.58% ± 1.49%	0.20% ± 1.28%

Table 6: Instrumentation overhead: Increase in runtime w.r.t. non-instrumented workload ± standard error.

11.2 Sensitivity

Histogram moving-average span. Due to large record sizes, some workload & fault combinations (e.g., `ddr` & disk-busy) yield request processing times up to 4 s. As client requests often synchronize (see § 12), metrics may reflect distinct request processing stages instead of aggregate behavior. For example, during a disk fault, the faulty server performs long, low-throughput storage operations while fault-free servers perform short, high-throughput operations. At 1 s resolution, these behaviors reflect asymmetrically in many metrics. While this feature results in high (79%) ITP rates, its presence in nearly all metrics results in high (10%) DFP rates as well. Furthermore, since the influence of this feature is dependent on workload and number of clients, it is not reliable, and therefore, it is important to perform metric smoothing.

However, "too much" smoothing eliminates medium-term variances, decreasing TP and increasing FP rates. With 9-point smoothing, DFP (11%) exceeds unsmoothed while DTP reduces by 11% to 58.3%. Therefore we chose 5-point smoothing to minimize IFP (2.4%) and DFP (6.7%) with a modest decrease in DTP (64.9%).

Anomalous window filtering. In histogram-based analysis, servers are flagged anomalous only if they demonstrate anomalies in k of the past $2k-1$ windows. This filtering reduces false-positives in the event of sporadic anomalous windows when no underlying fault is present. k in the range 3–7 exhibits a consistent 6% increase in ITP/DTP and a 1% decrease in IFP/DFP over the non-filtered case. For $k \geq 8$, the TP/FP rates decrease/increase again. We expect k's useful-range upper-bound to be a function of the time that faults manifest.

`cwnd` moving-average span. For `cwnd` analysis a moving average is performed on the time series to atten-

uate the effect of sporadic transmission timeouts. This enforces the condition that timeout events sustain for a reasonable time period, similar to anomalous window filtering. Spans in the range 5–31, with 31 the largest tested, exhibit a consistent 8% increase in ITP/DTP and a 1% decrease in IFP/DFP over the non-smoothed case.

WinSize & WinShift. Seven *WinSize*s of 32–128 with 16 sample steps, and seven *WinShift*s of 16–64 with 8 sample steps were tested to determine diagnosis influence. All *WinSize*s ≥ 48 and *WinShift*s ≥ 32 were comparable in performance (62–66% DTP, 6–9% DFP). Thus for sufficiently large values, diagnosis is not sensitive.

Histogram threshold scale factor. Histogram thresholds are scaled by a factor (currently 2x) to provide a cushion against secondary, minor fault manifestations (see § 10.1). At 1x, FP rates increase to 19%/23% IFP/DFP. 1.5x reduces this to 3%/8% IFP/DFP. On the range 2–4x ITP/DTP decreases from 70%/65% to 54%/48% as various metrics are masked, while IFP/DFP hold at 2%/7% as no additional misdiagnoses occur.

12 Experiences & Lessons

We describe some of our experiences, highlighting counterintuitive or unobvious issues that arose.

Heterogeneous Hardware. Clusters with heterogeneous hardware will exhibit performance characteristics that might violate our assumptions. Unfortunately, even supposedly homogeneous hardware (same make, model, etc.) can exhibit slightly different performance behaviors that impede diagnosis. These differences mostly manifest when the devices are stressed to performance limits (e.g., saturated disk or network).

Our approach can compensate for some deviations in hardware performance as long as our algorithm is trained for stressful workloads where these deviations manifest. The tradeoff, however, is that performance problems of lower severity (whose impact is less than normal deviations) may be masked. Additionally, there may be factors that are non-linear in influence. For example, buffer-cache thresholds are often set as a function of the amount of free memory in a system. Nodes with different memory configurations will have different caching semantics, with associated non-linear performance changes that cannot be easily accounted for during training.

Multiple Clients. Single- vs. multi-client workloads exhibit performance differences. In PVFS clusters with caching enabled, the buffer cache aggregates contiguous small writes for single-client workloads, considerably improving throughput. The buffer cache is not as effective with small writes in multi-client workloads, with the penalty due to interfering seeks reducing throughput and pushing disks to saturation.

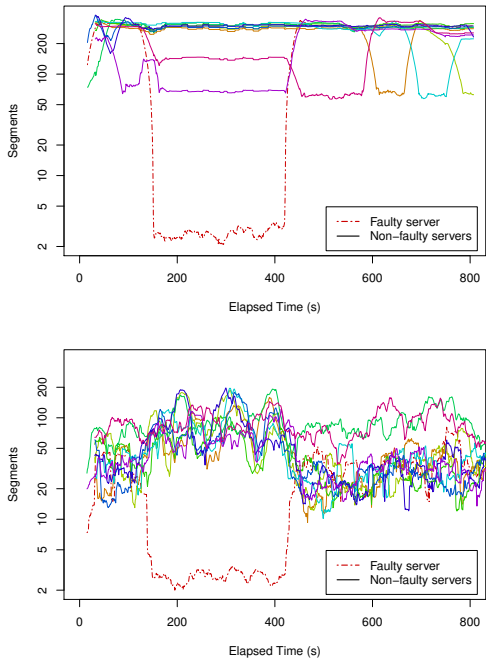

Figure 7: Single (top) and multiple (bottom) client `cwnd`s for `ddw` workloads with *receive-pktloss* faults.

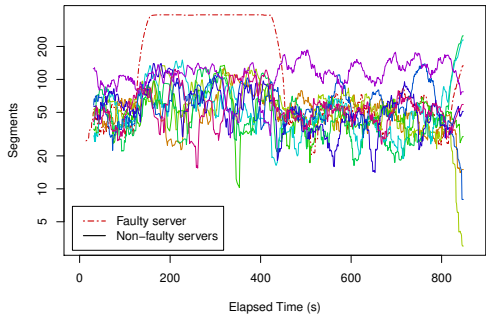

Figure 8: Disk-busy fault influence on faulty server's `cwnd` for `ddr` workload.

This also impacts network congestion (see Figure 7). Single-client write workloads create single-source bulk data transfers, with relatively little network congestion. This creates steady client `cwnd`s that deviate sharply during a fault. Multi-client write workloads create multi-source bulk data transfers, leading to interference, congestion and chaotic, widely varying `cwnd`s. While a faulty server's `cwnd`s are still distinguishable, this highlights the need to train on stressful workloads.

Cross-Resource Fault Influences. Faults can exhibit cross-metric influence on a single resource, e.g., a disk-hog creates increased throughput on the faulty disk, saturating that disk, increasing request queuing and latency.

Faults affecting one resource can manifest unintuitively in another resource's metrics. Consider a disk-busy fault's influence on the faulty server's `cwnd` for a

large read workload (see Figure 8). `cwnd` is updated only when a server is both sending and experiencing congestion; thus, `cwnd` does not capture the degree of network congestion when a server is *not* sending data. Under a disk-busy fault, (i) a single client would send requests to each server, (ii) the fault-free servers would respond quickly and then idle, and (iii) the faulty server would respond after a delayed disk-read request.

PVFS' lack of client read-ahead blocks clients on the faulty server's responses, effectively synchronizing clients. Bulk data transfers occur in phases (ii) and (iii). During phase (ii), all fault-free servers transmit, creating network congestion and chaotic `cwnd` values, whereas during phase (iii), only the faulty server transmits, experiencing almost no congestion and maintaining a stable, high `cwnd` value. Thus, the faulty server's `cwnd` is asymmetric w.r.t. the other servers, mistakenly indicating a network-related fault instead of a disk-busy fault.

We can address this by assigning greater weight to storage-metric anomalies over network-metric anomalies in our root-cause analysis (§ 10.2). With Lustre's client read-ahead, read calls are not as synchronized across clients, and this influence does not manifest as severely.

Metadata Request Heterogeneity. Our peer-similarity hypothesis does not apply to PVFS metadata servers. Specifically, since each PVFS directory entry is stored in a single server, server requests are unbalanced during path lookups, e.g., the server containing the directory "`/`" is involved in nearly all lookups, becoming a bottleneck.

We address this heterogeneity by training on the `postmark` metadata-heavy workload. Unbalanced metadata requests create a spread in network-throughput metrics for each server, contributing to a larger training threshold. If the request imbalance is significant, the resulting large threshold for network-throughput metrics will mask nearly all network-hog faults.

Buried ACKs. Read/write-network-hogs induce deviations in both receive and send network-throughput due to the network-hog's payload and associated acknowledgments. Since network-hog ACK packets are smaller than data packets, they can easily be "buried" in the network-throughput due to large-I/O traffic. Thus, network-hogs can appear to influence only one of `rxbyt` or `txbyt`, for read or write workloads, respectively.

`rxpck` and `txpck` metrics are immune to this effect, and can be used as alternatives for `rxbyt` and `txbyt` for network-hog diagnosis. Unfortunately, the non-homogeneous nature of metadata operations (in particular, `postmark`) result in `rxpck`/`txpck` fault manifestations being masked in most circumstances.

Delayed ACKs. In contradiction to Observation 5, a receive-(send-) packet-loss fault during a large-write (large-read) workload can cause a steady receive (send) network throughput on the faulty node and asymmetric decreases on non-faulty nodes. Since the receive (send) throughput is almost entirely comprised of ACKs, this phenomenon is the result of delayed ACK behavior.

Delayed ACKs reduce ACK traffic by acknowledging every other packet when packets are received in order, effectively halving the amount of ACK traffic that would otherwise be needed to acknowledge packets 1:1. During packet-loss, each out-of-order packet is acknowledged 1:1 resulting in an effective doubling of receive (send) throughput on the faulty server as compared to non-faulty nodes. Since the packet-loss fault itself results in, approximately, a halving of throughput, the overall behavior is a steady or slight increase in receive (sent) throughput on the faulty node during the fault period.

Network Metric Diagnosis Ambiguity. A single network metric is insufficient for diagnosis of network faults because of three properties of network throughput and congestion. First, *write-network-hog*s during write workloads create enough congestion to deviate the client `cwnd`; thus, `cwnd` is not an exclusive indicator of a packet-loss fault. Second, delayed ACKs contribute to packet-loss faults manifesting as network-throughput deviations, on `rxbyt` or `txbyt`; thus, the absence of a throughput deviation in the presence of a `cwnd` does not sufficiently diagnose all packet-loss faults. Third, buried ACKs contribute to network-hog faults manifesting in only one of `rxbyt` and `txbyt`, but not both; thus, the presence of both `rxbyt` and `txbyt` deviations does not sufficiently indicate all network-hog faults.

Thus, we disambiguate network faults in the third root-cause analysis step as follows. If both `rxbyt` and `txbyt` are asymmetric across servers, regardless of `cwnd`, a network-hog fault exists. If either `rxbyt` or `txbyt` is asymmetric, in the absence of `cwnd`, a network-hog fault exists. If `cwnd` is asymmetric regardless of either `rxbyt` or `txbyt` (but not both, due to the first rule above), then a packet-loss fault exists.

13 Related Work

Peer-comparison Approaches. Our previous work [14] utilizes a syscall-based approach to diagnosing performance problems in addition to propagated errors and crash/hang problems in PVFS. Currently, the performance metric approach described here is capable of more accurate diagnosis of performance problems with superior root-cause determination as compared to the syscall-based approach, although the syscall approach is capable of diagnosing non-performance problems in PVFS that would otherwise escape diagnosis here. The syscall-based approach also has a significantly higher worst-observed runtime overhead (≈65%) and per-server data volumes on the order of 1 MB/s, raising performance and

scalability concerns in larger deployments.

Ganesha [18], seeks to diagnose performance-related problems in Hadoop by classifying slave nodes, via clustering of performance metrics, into behavioral profiles which are then peer-compared to indict nodes behaving anomalously. While the node indictment methods are similar, our work peer-compares a limited set of performance metrics directly (without clustering), which enables us to attribute the affected metrics to a root-cause. In contrast, Ganesha is limited to identifying faulty nodes only, it does not perform root-cause analysis.

The closest non-authored work is Mirgorodskiy et al. [17], which localizes code-level problems by tracing function calls and peer comparing their execution times across nodes to identify anomalous nodes in an HPC cluster. As a debugging tool, it is designed to locate the specific functions where problems manifest in cluster software. The performance problems studied in our work tend to escape diagnosis with their technique as the problems manifest in increased time spent in the file servers' descriptor poll loop that is symmetric across faulty and fault-free nodes. Thus, our work aims to target the resource responsible for performance problems.

Metric Selection. Cohen et al. [8] uses a statistical approach to metric selection for problem diagnosis in large systems with many available metrics by identifying those with a high efficacy at diagnosing SLO violations. They achieve this by a summary and index of system history as expressed by the available metrics and by marking signatures of past histories as being indicative of a particular problem, which enables them to diagnose future occurrences. Our metric selection is expert-based, since in the absence of SLOs, we must determine which metrics reliably peer-compare to determine if a problem exists. We also select metrics based on semantic relevance, so that we can attribute asymmetries to behavioral indications of particular problems that hold across different clusters.

Message-based Problem Diagnosis. Many previous works have focused on path-based [1, 19, 3] and component-based [7, 16] approaches to problem diagnosis in Internet Services. Aguilera et al. [1] treats components in a distributed system as black-boxes, inferring paths by tracing RPC messages and detecting faults by identifying request flow paths with abnormally long latencies. Pip [19] traces causal request flows with tagged messages, which are checked against programmer-specified expectations. Pip identifies requests and specific lines of code as faulty when they violate these expectations. Magpie [3] uses expert knowledge of event orderings to trace causal request flows in a distributed system. Magpie then attributes system resource utilizations (e.g. memory, CPU) to individual requests and clusters them by their resource usage profiles to detect faulty requests. Pinpoint [7, 16] tags request flows through J2EE web-service systems, and, once a request is known to have failed, it identifies the responsible request processing components.

Each of the path- and component-based approaches rely on tracing of intercomponent messages (e.g., RPCs) as the primary means of instrumentation. This requires either modification of the messaging libraries (which, for parallel file systems is usually contained in server application code) or, at minimum, the ability to sniff messages and extract features from them. Unfortunately, the message interfaces used by parallel file systems are often proprietary and insufficiently documented, making such instrumentation difficult. Hence, our initial attempts to diagnose problems in parallel file systems specifically avoid message-level tracing by identifying anomalies through peer-comparison of global performance metrics.

While performance metrics are lightweight and easy to obtain, we believe that traces of component-level messages (i.e., client requests & responses) would serve as a rich source of behavioral information, and would prove beneficial in diagnosing problems with subtler manifestations. With the recent standardization of Parallel NFS [21] as a common interface for parallel storage, future adoption of this protocol would encourage investigation of message-based techniques in our problem diagnosis.

14 Future Work

We intend to improve our diagnosis algorithm by incorporating a ranking mechanism to account for secondary fault manifestations. Although our threshold selection is good at determining whether a fault exists at all in the cluster, if a fault presents in two metrics with significantly different degrees of manifestation, then our algorithm should place precedence on the metric with the greater manifestation instead of indicting one arbitrarily.

In addition, we intend to validate our diagnosis approach on a large HPC cluster with a significantly increased client/server ratio and real scientific workloads to demonstrate our diagnosis capability at scale. We intend to expand our problem coverage to include more complex sources of performance faults. Finally, we intend to expand our instrumentation to include additional black-box metrics as well as client request tracing.

15 Conclusion

We presented a black-box problem-diagnosis approach for performance faults in PVFS and Lustre. We have also revealed our (empirically-based) insights about PVFS's and Lustre's behavior with regard to performance faults, and have used these observations to motivate our analysis approach. Our fault-localization and root-cause analysis identifies both the faulty server and the resource at fault, for storage- and network-related problems.

Acknowledgements

We thank our shepherd, Gary Grider, for his comments that helped us to improve this paper. We also thank Rob Ross, Sam Lang, Phil Carns and Kevin Harms of Argonne National Laboratory for their insightful discussions on PVFS, instrumentation and troubleshooting, and anecdotes of problems in production deployments. This research was sponsored in part by NSF grants #CCF–0621508 and by ARO agreement DAAD19–02–1–0389.

References

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 74–89, Bolton Landing, NY, Oct. 2003.

[2] A. Babu. GlusterFS, Mar. 2009. `http://www.gluster.org/`.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pages 259–272, San Francisco, CA, Dec. 2004.

[4] D. Capps. IOzone filesystem benchmark, Oct. 2006. `http://www.iozone.org/`.

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

[6] P. H. Carns, W. B. Ligon, and R. B. R. andRajeev Thakur. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, Oct. 2000.

[7] M. Y. Chen, E. Kıcıman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, Bethesda, MD, June 2002.

[8] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, Brighton, UK, Oct. 2005.

[9] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, NY, Aug. 1991.

[10] J. Dean. Underneath the covers at Google: Current systems and future directions, May 2008.

[11] D. Gilbert. The Linux sg3_utils package, June 2008. `http://sg.danny.cz/sg/sg3_utils.html`.

[12] S. Godard. SYSSTAT utilities home page, Nov. 2008. `http://pagesperso-orange.fr/sebastien.godard/`.

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

[14] M. P. Kasick, K. A. Bare, E. E. Marinelli III, J. Tan, R. Gandhi, and P. Narasimhan. System-call based problem diagnosis for PVFS. In *Proceedings of the 5th Workshop on Hot Topics in System Dependability*, Lisbon, Portugal, June 2009.

[15] J. Katcher. PostMark: A new file system benchmark. Technical Report TR3022, Network Appliance, Inc., Oct. 1997.

[16] E. Kıcıman and A. Fox. Detecting application-level failures in component-based Internet services. *IEEE Transactions on Neural Networks*, 16(5):1027–1041, Sept. 2005.

[17] A. V. Mirgorodskiy, N. Maruyama, and B. P. Miller. Problem diagnosis in large-scale computing environments. In *Proceedings of the ACM/IEEE conference on Supercomputing*, Tampa, FL, Nov. 2006.

[18] X. Pan, J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan. Ganesha: Black-box diagnosis of mapreduce systems. In *Proceedings of the 2nd Workshop on Hot Topics in Measurement & Modeling of Computer Systems*, Seattle, WA, June 2009.

[19] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, , and A. Vahdat. Pip: Detecting the unexpected in distributed systems. In *Proceedings of the 3rd Conference on Networked Systems Design and Implementation*, San Jose, CA, May 2006.

[20] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *Proceedings of the 6th International Conference on Computer Vision*, pages 59–66, Bombay, India, Jan. 1998.

[21] S. Shepler, M. Eisler, and D. Noveck. NFS version 4 minor version 1. Internet-Draft, Dec. 2008.

[22] W. R. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001 (Proposed Standard), Jan. 1997.

[23] Sun Microsystems, Inc. Lustre file system: High-performance storage architecture and scalable cluster file system. White paper, Oct. 2008.

[24] The IEEE and The Open Group. dd, 2004. `http://www.opengroup.org/onlinepubs/009695399/utilities/dd.html`.

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

[26] S. A. Weil, S. A. Brandt, E. L. Miller, and D. D. E. Long. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pages 307–320, Seattle, WA, Nov. 2006.