Diagnosing Performance Problems in Parallel File Systems

Michael P. Kasick

Submitted in partial fulfillment of the
requirements for the degree of

Master of Science

Advisor: Priya Narasimhan

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania

May 2009

Abstract

This work describes and compares two black-box approaches, using syscall statistics and OS-level performance metrics, to automatically diagnose different performance problems in parallel file systems. Both approaches rely on peer-comparison diagnosis to compare statistical attributes of relevant metrics across servers in order to indict the culprit node. An observation-based checklist is developed to identify from the metrics affected the faulty resource, and it is demonstrated that this checklist applies commonly across stripe-based parallel file systems. These approaches are demonstrated for four realistic problems–disk-hog, disk-busy, network-hog, and packet-loss–injected into three different file-system benchmarks, dd, postmark, and IOzone, in PVFS and Lustre clusters.

Contents

Acknowledgments

I acknowledge my colleagues for their contributions to our problem diagnosis work: Keith Bare, Eugene Marinelli, and Jiaqi Tan for their contributions to *syscap*-based problem diagnosis and to compilation of related work; Rajeev Gandhi for his insights and contributions to data analysis; and Priya Narasimhan for her wordsmithing and incredible tact in conveying the technical details of this work. I also acknowledge Priya for serving as my graduate advisor and for her continued professional and personal support throughout my tenure as a student at Carnegie Mellon.

I especially 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—all of which are instrumental in motivating this work; John Wilkes of HP Labs for his insights on compelling errors for study; and Garth Gibson for his feedback on the *syscap* work.

I acknowledge Greg Ganger, both for serving as the second reader of this thesis, and also for his service as Director of the Parallel Data Lab, an organization through which I've had priceless opportunities to interact with and receive feedback from industry practitioners concerning this work.

Special thanks are due to my parents and family, who have always provided unparalleled support and encouragement for my education. Finally, I would like to thank my wife-to-be, for her unconditional love and support, and for her constant encouragement in the pursuit of my dreams.

Chapter 1

Introduction

Performance problems are inevitable in cluster computing, with root causes varying widely, from hardware failures to software bugs. For instance, Google reported the variety of performance problems that occurred in the first year of a cluster's operation [15]: 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 trace are not necessarily 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. This work targets problem diagnosis (identifying the culprit node that is the source of the problem) in parallel file-systems used for high-performance cluster computing (HPC).

Large scientific applications are typically I/O-intensive and require underlying file systems that can support high-bandwidth concurrent writes. The Parallel Virtual File System (PVFS) [10] and Lustre [35] are both open-source, parallel file-systems that aim to provide such applications with high-speed data read/write 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. Both PVFS and Lustre support the UNIX I/O interface and allows existing UNIX I/O programs to access files without recompilation. To facilitate parallel access to a file, PVFS and Lustre distributes (or "stripes") that file across multiple disks located on physically distinct I/O servers.

File systems can experience performance problems that can be hard to diagnose and isolate. Performance problems can arise from different system layers, e.g., bugs in the application, resource exhaustion misconfigurations of protocols, network congestion, etc. PVFS and Lustre are largely used within the HPC community, where performance is highly relevant. Thus, while diagnosing performance problems is important in software systems in general to improve task-completion times, it is even more important in long-running jobs in HPC environments, where the effects of performance problems can be magnified due to computations exhibiting long durations and being performed at large scales. Current diagnosis of PVFS problems involve the manual analysis of client/server debug logs that record PVFS operations through code-level print statements. Such white-box problem-diagnosis is fairly expensive in terms of runtime overheads, and also requires code-level instrumentation and expert knowledge.

This work focuses on automatically diagnosing different performance problems in parallel file systems by identifying, gathering and analyzing black-box performance metrics on every node in the system. It

describes and compares two implementations of this black-box diagnosis strategy, one that distills statistics from syscall event streams, and another that samples operating system performance metrics. Central to the problem-diagnosis strategy is the hypothesis (borne out later by observations of PVFS's and Lustre's behavior) that *innocent (i.e., fault-free) I/O servers follow similar trends in their disk and network performance metrics, and that a culprit I/O server appears markedly different in comparison*. A similar hypothesis follows for the metadata servers. Based on these hypotheses, statistical peer-comparison approach is developed that automatically singles out the culprit I/O (or metadata) server through the temporal correlation of the histograms and other statistical attributes of gathered performance data across the I/O (or metadata) servers in a parallel file system cluster.

This diagnosis approach is demonstrated for four realistic problems injected into three different file-system benchmarks in PVFS and Lustre clusters. Interestingly, but unsurprisingly enough, the peer-comparison diagnostic strategy indicts the culprit node correctly even in the face of workload changes, which usually present a source of false positives for most statistical problem-diagnosis techniques. Experience in diagnosing PVFS and Lustre is also discussed, particularly in terms of the value of metrics such as disk throughput and disk latency for diagnosis.

1.1 Problem Statement

This research is motivated by the following questions: (i) *can one diagnose the culprit server in the face of a performance problem in a parallel file system*, (ii) *if so, can one determine which server is affected and what problem is causing the effect?*, and (iii) *what instrumentation sources are available that enables one to do so?* The following goals (and non-goals) are sought (or not sought) in addressing these research questions.

1.1.1 Goals

The following goals are sought to be satisfied in answering the research questions:

- *Application-transparency.* Applications using PVFS/Lustre should not require any modification. The approach should be independent of PVFS/Lustre's operation.
- *Minimize false-positive rate.* The approach should be able to differentiate between anomalous behavior and legitimate changes in behavior (e.g., workload changes due to increased request rate).
- *Minimize instrumentation overhead.* Overheads for instrumentation and analysis should not adversely impact PVFS/Lustre's operation.
- *Problem coverage.* Problem coverage is motivated by the anecdotal experience [9] of problems that have occurred in a production parallel file system deployment (see § 2.2).

1.1.2 Non-Goals

In answering the research questions, it is currently not sought to satisfy the following goals:

- *Code-level debugging.* The approach aims for coarse-grained problem diagnosis by identifying the culprit server. Where possible, it is sought to identify which resource is under contention. However, it is not currently aimed for finegrained problem diagnosis that would identify the underlying bug or offending lines of PVFS/Lustre code.

- *Pessimistic workloads.* The peer-comparison diagnosis strategy relies on I/O servers exhibiting similar request patterns. In non-replicated parallel file systems, the request pattern for most workloads is similar across all servers—client requests are either large enough to be striped across all servers, or random enough to result in random distribution of access requests. However, some workloads (e.g., overwriting the same portion of a file, or only writing stripe-unit-sized records to every stripe-count record offset) make requests distributed to only a subset, possibly one, of the servers. This work does not aim to support these workloads, which indeed may generate false positives.

- *Diagnose non-peers.* The diagnosis strategy is fundamentally incapable of diagnosing performance problems on non-peer nodes. Thus it is not aimed to diagnose problems on such non-peered nodes (such as Lustre's single metadata server).

1.2 Thesis Statement

This thesis focuses on problem diagnosis in distributed systems. Specifically, the thesis statement is:

> *Through the collection of black-box data such as syscall statistics and OS-level performance metrics, it is possible to automatically and transparently indict the culprit node and diagnose the root cause of a performance problem in parallel file system clusters.*

1.2.1 Hypotheses

It is hypothesized that, under a performance fault in a PVFS or Lustre cluster, performance metrics should exhibit observable anomalous behavior on the culprit servers. Additionally, with the knowledge of PVFS/Lustre's overall operation, it is hypothesized 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.

1.2.2 Validation

These hypotheses are validated through the design and implementation of two agents to collect syscall statistics and OS-level performance metrics as well as two analysis strategies to diagnose performance problems using the collected data. These analysis strategies are evaluated by injecting the four performance problems motivated by anecdotal evidence—disk-hog, disk-busy, network-hog, and packet-loss—into three different file-system benchmarks, dd, postmark, and IOzone. This method of evaluation is performed, for OS-level performance metrics, on both PVFS and Lustre clusters. Since the Lustre client and server are both implemented purely in kernel space, it is not possible to apply the syscall approach to it. Thus, the syscall statistics approach is evaluated against PVFS only.

1.2.3 Assumptions

It is assumed that a majority of the I/O servers exhibit fault-free behavior. It is also assumed that all peer server nodes are *similar*, with identical software configuration including data striping parameters, with an equal number of storage targets (of same size) per server,[1] the same amount of memory,[2] and the same class and speed-rating of network interfaces. Nodes do not need to be strictly homogeneous, but significant deviations in performance characteristic of storage and network devices may mask performance faults of lower severity. It is assumed that the physical clocks on the various nodes are synchronized (e.g., via NTP) so that one can extract time-stamped performance metrics that can be temporally correlated across these nodes.

1.3 Related Work

Past work on failure diagnosis in distributed systems has focused mainly on Internet services [2, 12, 13, 11, 6, 23, 29, 1], with some work examining debugging performance problems in high-performance computing environments [26]. This work is different in targeting HPC file-systems instead of the deployed workload application.

The relevant aspects of past work on fault diagnosis in distributed systems are in their approaches to diagnosing faults. These can be largely classified into two categories: path-based approaches [2, 29, 6], and component-based approaches [13, 11, 12, 23].

Aguilera et al.'s work [2] 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 [29] 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 [6] 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. Magpie then detects faulty requests which identifying those whose resource usage profiles fell outside of all clusters.

Pinpoint [12, 23] tags request flows through J2EE web-service systems, and once a request is known to have failed, it identifies the responsible request processing components. This differs from the work by Aguilera, Magpie, and Pip, in that the fault localization is at the component level and not path-based. Cohen et al. [13] addressed the issue of metric selection for fault diagnosis in large systems with many available metrics by identifying metrics with a high efficacy at identifying failed requests. This is achieved by a summary and index of system history as expressed by available metrics and by marking signatures of past histories as being indicative of a particular fault, which enables them to diagnose future occurrences. Agarwal et al.'s work focuses on detecting changepoints in the specific metrics that reflect patterns of behavior as indicators of faults based on a series of rules [1].

[1] Ensures that Lustre objects are selected round-robin and that request load on servers is approximately equal

[2] Relevant only to the extent to which it alters caching behavior.

1.3.1 Performance Modeling and Prediction

Closely related to failure diagnosis is the problem of performance modeling and prediction—information that can be used as input to failure diagnosis algorithms. Minerva [3] automatically makes design choices for storage systems with accurate performance predictions. Mesnier et al. [24] model the performance of storage devices using by comparing performance across pairs of devices to predict their throughput and bandwidth.

1.3.2 Instrumentation and Trace Systems

Tracing and instrumentation generate system views that are useful for diagnosis. File system-specific tracing mechanisms include Stardust [37], which traces causal request flows through a distributed storage system, and TraceFS [4], which uses a thin file-system interpositioned between the Linux VFS layer and the underlying file-system to provide operation traces at multiple granularities. PVFS is positioned at the High-Performance Computing (HPC) community, and is commonly used on large-scale clusters for scientific applications, such as those using the Message Passing Interface (MPI). Performance tools for HPC environments include TAU [31] and Paradyn Parallel Performance Tools [25]. For low-level OS instrumentation, Chopstix [7] uses sketches to approximately track a large number of OS events with very low overhead. These events may then be reconstructed offline for analysis.

1.3.3 Syscall-based Approaches

Shen [30] and OSprof [20] 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 captures and analyzes distributions of syscall latencies under different system conditions to identify code paths that exhibit latency peaks. The statistical analysis presented here 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.

1.3.4 Failures in Storage Systems

Other related work includes studies of storage system failures. Jiang et al. [19] conclude from storage logs of 1,800,000 disks deployed at Network Appliance customer sites that disk failures contributed to 20-55% of storage subsystem failures, while other causes included physical interconnects and protocol stacks that led to disk replacement. This finding motivates failure diagnosis at the level of the software stack, as this can point out causes of failure due to software rather than hardware to avoid costly hardware replacements. Prabhakaran et al. [27] analyze failures in journaling file-systems by building models of file system behavior. Work to increase the fault-tolerance of file-systems takes failures into consideration. IRON File Systems [28] re-examines the approach of traditional file-systems towards error handling, analyzes file-system failure policies, and boosts the end-to-end robustness of the *ext3* file-system.

Chapter 2

Background

2.1 PVFS & Lustre Architectures

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 2.1. The PVFS server consists of a single monolithic user-space daemon that may act in either or both metadata and I/O server roles. Historically I/O and metadata services were ran on separate nodes, but recent versions of PVFS encourage running both services on all nodes to enhance performance.

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 in a fashion similar to other remote file-systems like NFS.

PVFS's key characteristic is that 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 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 configurable striping of file data across one or more object storage targets. In the default configuration, file data is stored on a single target. The `stripe_count` parameter may be set on a

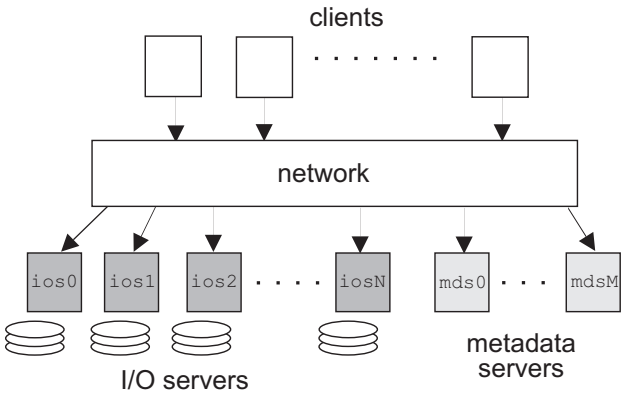

Figure 2.1: Architecture of parallel file systems, showing the I/O servers and the metadata servers.

per-file, directory, or file system basis to specify the number of object storage targets that file data is striped over. The stripe `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). The `stripe_offset` parameter determines on which object storage target the first stripe segment resides, with -1 indicating that first segments should be allocated round-robin.

2.2 Anecdotal Motivation

The faults studied in this work are motivated by the PVFS developers' anecdotal experience [9] 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 disk/network problems account for approximately 50%/50% of performance issues [9]. 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 [9]. 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 disk 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 [18], inadvertently creating disk contention that degrades the throughput of a disk array [38]. The *disk-busy* injected problem (§ 4.1) seeks to emulate this manifestation. Another possible root-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) [9].

Network problems primarily manifest in packet-loss errors, which is reported to be the "most frustrating" [sic] to diagnose [9]. 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 [9]. 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 [9].

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 [9]. In contrast, performance problems in read-intensive workloads tend to 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) [9].

Chapter 3

Instrumentation

3.1 Syscall Instrumentation

System-call instrumentation includes recording the syscall event-stream of a particular process (e.g., application, PVFS client, PVFS server) and extracting the metrics relevant for analysis.

3.1.1 *Syscap*: Black-box Syscall Capture

The `syscap` tool was developed for the purpose of instrumenting syscalls. Unlike `strace`, `syscap` produces a regular, structured output that is amenable to machine analysis. In contrast, `strace`'s irregular, truncated output is intended primarily to facilitate manual debugging.

The `syscap` tool uses `ptrace` (a user-space Linux API for syscall interception and modification, *without* changes to the monitored process) to trace three kinds of events: syscalls, signals, and `ptrace` events. `ptrace` events are notifications of important process-lifecycle events (fork, clone, exec, exit) that are delivered to `syscap` immediately, prior to the kernel executing the associated operation.

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 (that identifies events in order of completion), a sequence number (that identifies events in order of start), timestamp (at the start of the event), light-weight process ID, syscall/signal/event number, syscall register arguments (a total of six in all), syscall result, and syscall wall-clock service time.

In addition to the `sclog`, `syscap` produces a file-descriptor update-log (`fdlog`) consisting of records that describe updates to traced processes' file-descriptor tables. Record fields consist of: a record number (to link an update to the generating `sclog` event), the file-descriptor number, and the target file name. Records are generated when a syscall is encountered that changes the open-file table (e.g., `open`, `close`, `accept`, `connect`, etc.). The target file name is determined by reading the target of the `/proc/pid/fd/#` symlink. For regular files, the target is simply the file name referenced by the symlink. For anonymous "files" (unnamed pipes, network sockets, etc.), a special identifier is maintained that possibly includes other discriminating characteristics of that entity. For example, TCP-socket "files" include the end-point IP address and port number, enabling the mapping of a socket to a specific client-server link.

To complement `syscap`, a set of metric-extraction utilities were developed to parse the `sclog` and

fdlog files. These utilities invoke experiment-specific callback functions when processing specific syscalls, signals, and file-descriptor updates, and extract the metrics relevant for that experiment's analysis. To extract syscall statistics for diagnosing performance problems, these utilities look for file-descriptors associated with files (server file-objects) stored in the PVFS storage-space as well as TCP sockets between I/O servers and clients. On observing the creation of such a file-descriptor, these utilities track all syscalls that subsequently use the file-descriptor, and extract the service times of all associated read and write syscalls.

3.1.2 Metrics of Interest

In 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.

Since PVFS uses non-blocking reads, the amount of time needed to transfer an entire request/response over the network is estimated by the time (timestamp) elapsed between the read of the first received packet of a request, and the time (timestamp + service time) of the read of the last received packet of that request. Requests are demarcated by observing the number of bytes intended to be read as well as the number of bytes actually returned by the read call. On receiving a request, PVFS issues a read of the full request-size. Each additional read call specifies the remaining request-size (number of bytes not yet received). A request is considered to have been received when the request-size equals the number of bytes received. Note that, implicitly, a request must require at least two reads in order to estimate the transmission time. Also, while this transmission time is merely an estimate of the actual transmission time (because it does not factor in the amount of time to send the request's first packet), it grows linearly with the actual transmission time.

3.2 /proc Instrumentation

In Linux, OS-level performance metrics are made available as text files in the /proc pseudo file system. Thus, /proc instrumentation is the mechanism by which OS-level performance metrics are sampled and collected. Most /proc data is collected via sysstat 7.0.0's sadc program [17]. sadc is used to periodically gather disk-, and network-related performance metrics at a sampling interval of one second. Although sysstat supports the collection of many types of metrics (e.g., CPU usage), because this work is primarily concerned with performance problems due to storage and network resources, sadc's use is restricted to sampling to disk- and network-related metrics.

Disk-related metrics: Sysstat provides both disk-throughput (tps, rd_sec, wr_sec) and disk-latency (await, svctm) metrics for storage. Specifically, the following metrics are collected:

- Transfers per second (tps). Number of I/O requests made to the disk per second; includes both read and write requests.
- Reads per second (rd_sec). Number of sectors read from the disk per second.

- Writes per second (`wr_sec`). Number of sectors written to the disk per second.
- Average request size (`avgrq-sz`). Average size (in sectors) of disk I/O requests.
- Average queue length (`avgqu-sz`). Average number of queued disk I/O requests; this is a generally a low integer (0-2) when the disk is under-utilized, and increases to ≈100 as the disk utilization saturates.
- Average wait time (`await`). Average time (in milliseconds) that a request waits to complete; includes queuing delay and service time.
- Service time (`svctm`). Average service time (in milliseconds) of I/O requests; this is the pure disk-servicing time, and does not include any queuing delay.
- Bandwidth utilization (`%util`). Percentage of CPU time in which I/O requests are made to the disk.

Network-related metrics: Sysstat provides the following network-throughput metrics:

- Packets received per second (`rxpck`).
- Packets transmitted per second (`txpck`).
- Bytes received per second (`rxbyt`).
- Bytes transmitted per second (`txbyt`).

Unfortunately sysstat provides only throughput data for network performance metrics. To obtain network-congestion data as well, the contents of `/proc/net/tcp` is sampled, on both clients and servers, once every second. `/proc/net/tcp` captures per-connection metrics and, in particular, TCP congestion-control data [34] from which network congestion can be deduced. Specifically, `/proc/net/tcp` provides the sending congestion window (`cwnd`) metric—the number of segments allowed to be sent outstanding without acknowledgment.

Chapter 4

Diagnostic Approach

Parallel file system workloads typically have high bandwidth requirements. For example, scientific applications may write from many client nodes to a single very large file. Parallel file systems support these high bandwidth workloads by distributing the workload across many server machines to achieve greater throughputs than a single server could. In this case, the hardware resources that are most likely to pose as bottlenecks are network and storage. Fail-stop performance problems usually result in an outright crash of a server, making it relatively straightforward to indict the culprit server. This work targets non-fail-stop disk and network performance problems as this class of hardware performance-faults can reduce server performance by at least an order of magnitude, without escalating into an outright server crash.

4.1 Failures of Interest

This work defines a performance fault as a problem that has an observable negative effect on I/O throughput. There are basically three resources—CPU, disk, network—being contended for that are likely to result in 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, focus is on the remaining two resources, disk and network, that are likely to pose performance bottlenecks.

Motivated by anecdotal experience of problems encountered in production deployments (§ 2.2), this work separates failures involving disk and network resources into two classes. The first class is *hog* faults, where a rogue process on the monitored file-servers creates 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, this work studies the diagnosis of four problems—disk-hog, disk-busy, network-hog, packet-loss (network-busy).

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; this is viewed 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 result of packet corruption and packet-loss from a failing NIC.

4.2 Observations of Parallel File System Behavior

This section highlights some (empirically-based) observations of PVFS's and Lustre's behavior that, it is believed, generally characterize the behavior of striped-based parallel file systems.[1]

[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[2] 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. This is observed for all three target benchmarks, dd, PostMark and IOzone. For metadata-intensive workloads,[3] it is expected that metadata servers also track each other in proportional magnitudes of throughput and latency.

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

When a client I/O syscall involves requests to multiple I/O servers, the client has to wait for all of these servers to respond before proceeding onto the next syscall.[4] Thus, the cluster's performance, as perceived by the client, is constrained by the performance of the slowest server. This is referred to as the *bottlenecking condition*.

When one of the servers experiences a performance fault, that server's per-request service-time increases. Because the client blocks on the syscall until all server responses are received, the client's syscall-service time also increases. This leads to slower application progress and fewer requests/sec from the client. The overall effect is a proportional decrease in throughput on all I/O servers.

[1] Preliminary investigation of two other parallel filesystems, GlusterFS [5] and Ceph [39], also exhibit the behavior observed in PVFS and Lustre, suggesting that this diagnosis approach would apply to them as well.

[2] Pessimistic 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. Examples include a workload that constantly rewrites the same *stripe_size* chunk of a file, or one that writes to the *stripe_size* chunks of a file that are stored on the same I/O server. Performance faults under such workloads would escape diagnosis.

[3] The metadata servers—at least in PVFS—will never be exactly equivalent since metadata operations often involve path lookups. With the root directory being entirely located on a single metadata server, that particular server should see the most activity as every path lookup requires looking though "/" in addition to later directories. Still, one expects throughput/latency increases and decreases to be reflected across all metadata servers.

[4] 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.

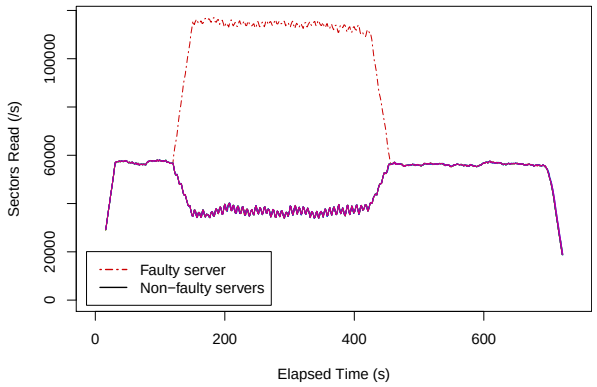

Figure 4.1: Peer-divergence of `rd_sec` for `iozoner` workload with *disk-hog* fault.

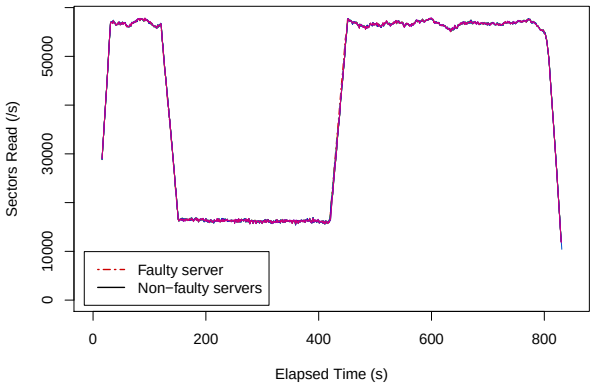

Figure 4.2: No divergence of `rd_sec` for `iozoner` workload with *disk-busy* fault.

[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 (thus, no server inter-dependencies), a performance fault on one I/O server will not negatively influence latency (per-request service-time) of the other servers. If these servers were previously highly-loaded, one expects latency to possibly improve (due to potentially decreased resource contention). If the servers were only marginally loaded, one would expect no change.

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

A disk-hog or network-hog fault is due to a third-party that creates additional I/O traffic that is observed as increased disk- or network-throughput. The additional I/O traffic creates resource contention that ultimately manifests as a decrease in file server throughput on all servers (leading to the bottlenecking condition of observation 2). Here, disk-throughput is measured in terms of the metrics `rd_sec` and `wr_sec` (or alternatively `tps`). Network-throughput is measured in terms of `rxbyt` and `txbyt` (or `rxpck` and `txpck`). Thus, disk- and network-hog faults can be localized to the culprit I/O server by looking for asymmetry (or *peer-divergence*) in the disk- and network-throughput metrics, respectively. This is seen in Figure 4.1.

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

For disk-busy (packet-loss) faults, there is no asymmetry in disk (network) throughputs, respectively, across I/O servers (because there is no other process that creates observable throughput, and the server daemon has the same throughput on all of the nodes). Instead, there is a uniform decrease in the disk (network) throughput metrics across all servers. Thus, disk-busy and packet-loss faults cannot be localized to the culprit I/O server by looking for peer-divergence in the disk-and network-throughput metrics, respectively. This is seen in Figure 4.2.

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

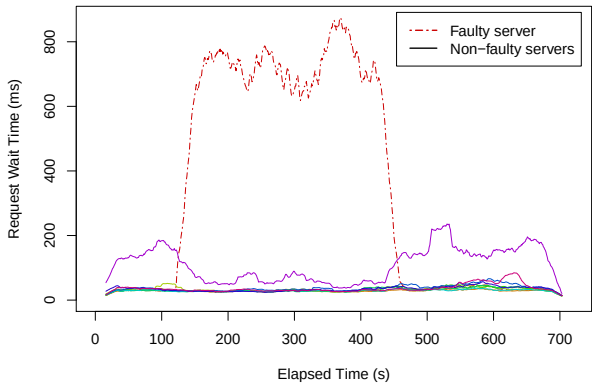

Figure 4.3: Peer-divergence of `await` for `ddr` workload with *disk-hog* fault.

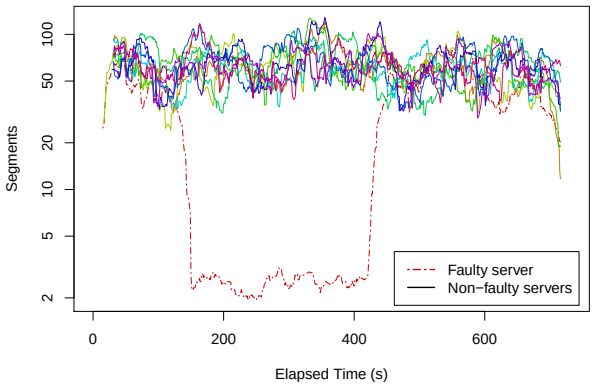

Figure 4.4: Peer-divergence of `cwnd` for `ddw` workload with *receive-pktloss* fault.

Disk-latency metrics (`await`, `svctm`) exhibit different responses to network and disk faults. For disk-busy and disk-hog faults, `await`, `avgqu-sz` and `%util` all increase on the faulty I/O server as the disk's responsiveness decreases and requests start to backlog. The increased `await` on the faulty server results in an increased server response-time, effectively making the client wait longer before it can issue its next request. The additional delay experienced by the client reduces its I/O throughput, resulting in the fault-free servers having increased idle time. Thus, the `await` and `%util` metrics decrease on the fault-free I/O servers, enabling a peer-comparison-based approach to diagnose disk-hog and disk-busy faults. This is seen in Figure 4.3.

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

The `cwnd` metric indicates the number of segments that the TCP congestion-control algorithm allows to be outstanding (i.e., without acknowledgment) on the wire for a given client-server connection. The goal of 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 reduced to half. If recovery takes longer than retransmission timeout, `cwnd` is reduced to one segment.

When nodes are transmitting data, their `cwnds` metrics either stabilize at high (≈100) values or oscillate (between ≈10–100) as congestion is observed on the network. However, during both network-hog and packet-loss experiments, `cwnds` of connections to the faulty server drops by several orders of magnitude to single-digit values and hold steady until the fault is removed, at which time the congestion window is allowed to open again. Observation of these sustained drops compared to non-faulty I/O servers enables peer-comparison diagnosis for both network faults. This is seen in Figure 4.4.

4.3 Discussion on Metrics

Comments on disk-throughput metrics: There is a notable relationship between the disk- and network-throughput metrics: `tps` × `avgrq-sz` = `rd_sec` + `wr_sec`. This relationship is partially to blame for the

reliability of disk-throughput metrics in diagnosis. Of the metrics, `rd_sec` and `wr_sec` are more accurate in capturing real disk activity, and are strongly correlated across peer I/O servers. The peer relationship of `tps` across servers, and that of `avgrq-sz` across servers, are not as strong, as a lower transfer rate can be compensated by making larger-sized requests. Occasionally, one sees the `tps` and `avgrq-sz` metrics fluctuate on one or more servers due to behavioral differences in the flush daemon or differing construction of the scatter gather lists used for device DMA—in the extreme, this behavior can result in wrongly indicting a fault-free node as the culprit. Thus, `rd_sec` and `wr_sec` are the preferred throughput metrics for this work.

Comments on `svctm`: The impact of disk faults on `svctm` is perhaps a bit counterintuitive. Essentially, there are three factors that influence the service time of a particular request: the cost of locating the request's starting sector (seek time and rotational delay), the cost of media-transfer time, and the cost of a potential reread/rewrite in the event of a read/write error. Normally, the disk is servicing other (potentially unobservable) requests that might result in increased seek time, in the event of a disk fault. Thus, for an unchanged `avgrq-sz`, `svctm` will increase on the faulty I/O server, as compared to its peer I/O servers. However, in the case of a disk-hog fault, the hog process might be issuing requests of different size than PVFS or the flush daemon—if the hog process' disk requests are smaller in size, then, there is a decrease in media-transfer time that might result in a decrease of `svctm` on the faulty I/O server, as compared to its peer I/O servers.

Other metrics: While performance faults might manifest on other metrics (such as CPU usage, context-switch rate, paging rate, etc.), these secondary manifestations are due to the overall reduction in I/O throughput during the faulty period. As these metrics report "nothing new", they are not factored into diagnosis and root-cause analysis.

Chapter 5

Experimental Set-Up

5.1 Testbed

Experiments are performed 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/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. PVFS experiments consist of 10 combined I/O and metadata servers and 10 clients, while Lustre experiments consist of 10 object storage (I/O) servers with a single object storage target each, a single (dedicated) metadata server, and 10 clients.

For these experiments PVFS 2.8.0 is used in the default server (`pvfs2-genconfig` generated) configuration with two modifications. First, the Direct I/O method (`TroveMethod directio`) is used to bypass the Linux buffer cache for PVFS I/O server storage as it ensures syscall metrics reflect disk service times instead of cache hits, and also because its use confounds diagnosis with OS-level performance metrics (see § 8.2 for the difficulties introduced by cache behavior). Second, the Flow buffer size (`FlowBufferSizeBytes`) is increased to 4 MB (from 256 kB) to allow larger bulk data transfers and enable more efficient disk usage.

The PVFS kernel client is also patched 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. The PVFS kernel client is then invoked with 64 MB request buffers (`desc-size` parameter) in order to make 4 MB data transfers to each of the 10 I/O servers.

For Lustre experiments the etch backport of the Lustre 1.6.6 Debian packages are used in the default 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`; no time synchronization is performed during experiments to avoid data discrepancies due to unexpected time adjustments. For PVFS experiments, a new *ext2* file system is created on the dedicated storage disk of each server node after nodes are rebooted. Then, in both PVFS and

Lustre experiments, the PVFS/Lustre `mkfs` routine is invoked to set up the storage subsystem.[1] Once the servers are initialized and the client is mounted, the monitoring agents are initialized and capture metrics to a local (non-storage dedicated) disk, `sync` is performed, followed by a 15-second sleep, and the experiment benchmark is run.

The experiment benchmark runs for 120 seconds in fault-free mode, 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 is ideally timed to run 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 has been removed in order to determine if there are any delayed effects.

5.2 Workloads

The five experiment workloads used are derived from three experiment benchmarks: `dd`, PostMark, and IOzone. For all workloads, 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` [36] performs a constant-rate, constant-workload large-file read/write from/to disk. It is the simplest large-file benchmark to run, and it was chosen in order to analyze and understand the response of the system prior to running more complicated workloads. Although not necessarily representing the behavior of any specific application, `dd` models the behavior of scientific computing workloads with constant data-write rates, an important class of applications that PVFS aims to support.

The next two workloads, `iozonew` and `iozoner`, consist of the same common file-system benchmark, IOzone v3.283 [8]. `iozonew` is run in write/rewrite mode and `iozoner` in read/reread mode. Each IOzone benchmark uses only a single test to ensure that faults are injected with the same workload characteristics across each redundant experiment, making results comparable. 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 the benchmark.

The fifth benchmark is PostMark v1.51 [22]. 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).

5.2.1 Configurations of Workloads

For the `ddw` workload, a 17 GB file is used with a record-size of 40 MB for PVFS, and a 30 GB file is used with a record-size of 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 was 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, it was found that varying record-size does not significantly alter Lustre read or write performance. For `ddr` a

[1] Lustre's `mkfs` automatically reformats the storage partition using its custom *ldiskfs* file system, hence no separate formatting step is performed.

27 GB file is used with a record-size of 40 MB for PVFS, and a 30 GB file is used with a record-size of 10 MB for Lustre (same as `ddw`).

For both the `iozonew` and `iozoner` workloads, an 8 GB file is used with a record-size of 16 MB (the largest record-size that IOzone supports without modification) for PVFS. For Lustre a 9 GB file is used with a record-size of 10 MB for `iozonew`, and a 16 GB file is used with the same record-size for `iozoner`. The `postmark` workload uses the default configuration with 16,000 transactions for PVFS and 53,000 transactions for Lustre to give a sufficiently long-running benchmark.

5.3 Fault Injection

In fault-induced experiments, a single fault is injected at a time into one of the I/O servers with the intent of inducing a degraded performance for either network or disk resources, causing the system to bottleneck on that resource.

A *disk-hog* problem is simulated by running a `dd` process, reading 256 MB blocks (using direct I/O) from an unused partition on one of the storage disks.

A *disk-busy* problem is simulated by running an `sgm_dd` process (from the Linux sg3_utils package [16]), which issues low-level SCSI I/O (`SG_IO`) commands via the Linux SCSI Generic (`sg`) driver, to read 1 MB blocks from the same unused partition on one of the storage disks. By using SCSI I/O commands via the `sg` driver the SCSI Disk (`sd`) driver is bypassed, hence these commands are not reflected in the storage disk's throughput performance metrics, and are thus, unobservable[2] as they would be if they were made by a third-party server to a shared storage device.

A network-hog problem is simulated either by having a third-party open a TCP connection to a listening port on one of the PVFS I/O servers and sending zeros to it (*write-network-hog*) or by having the I/O server send zeros to the third-party (*read-network-hog*).

A server receive-packet-loss (*receive-pktloss*) problem is simulated by a netfilter firewall rule that probabilistically matches packets received at one of the I/O servers drops them with probability 5%. A server send-packet-loss (*send-pktloss*) problem is simulated by a firewall rule on all clients that matches packets incoming from a single server (this has the same effect as placing the firewall rule on the server output, except that in the latter case, netfilter returns an error to the sending application instead of silently dropping packets).

[2]The term "unobservable" refers to the fact that the workload is not directly observed as transactions (`tps`) or throughput (`rd_sec`/`wr_sec`). However one can infer the workload based on its impact on latency.

Chapter 6

Diagnosis Algorithm

Both the syscall statistic- and OS performance metric-based approaches target disk and network performance problems for diagnosis in parallel file system clusters. Both algorithms rely on the hypothesis that, under fault-free conditions, different I/O servers have similar average behavior (because of an equitable distribution of client requests across them). Both algorithms also exploit the expected asymmetrical (i.e., peer-divergent) behavior of faulty I/O servers, as compared to their fault-free counterparts, to identify the faulty servers.

6.1 Syscall-Based Diagnosis

This approach diagnoses disk and network performance problems by statistically comparing the time-series of syscall durations across all the I/O servers in a PVFS cluster. As previously stated, this algorithm relies on the 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, such as storage disks with different performance profiles, can cause some of the server disks to be request saturated). Such heterogeneity is accounted for through a *fault-free training phase*. The underlying hypothesis is that while a saturated server's syscall service times 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, time-series of average syscall service times is generated at each server by dividing the sum syscall duration by the number of syscalls at 1-second intervals. The 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, a representative value is computed (currently, the median of the syscall service times across all the non-faulty/non-saturated servers is used) 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, the maximum deviation of a server's syscall service times is determined from the representative (median) value under fault-free conditions. This phase also determines which servers are likely to be saturated under fault-free conditions. The `ddw` and `ddr` workloads are currently used, 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

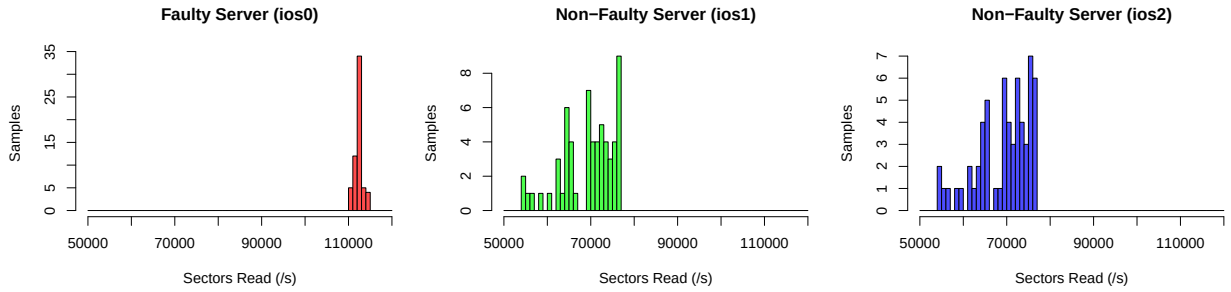

Figure 6.1: Histograms of `rd_sec` (`ddr` with *disk-hog* fault) for one faulty and two non-faulty servers.

deviation. For each server, the maximum deviation of its service times from the median value is detected and the maximum deviation is used to choose the 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.

6.2 Performance Metric-Based Diagnosis

This diagnosis technique analyzes OS-level performance metrics to diagnose faulty servers in both PVFS and Lustre clusters in two phases. The first phase of the peer-comparison diagnostic algorithm identifies the culprit I/O server for the faults targeted.

6.2.1 Identifying the Culprit Server

In development of the peer-comparison diagnostic algorithm, several statistical properties were considered (e.g., the mean of the metric value, the variance of the metric value) as candidates for comparison across I/O servers, but ultimately the probability distribution function (PDF) of each metric value was chosen as the best property to compare. The reason is that the PDF captures many of the other statistical properties, such as the mean and variance. Figure 6.1 shows how a metric's histograms/PDFs differ between the culprit server and the other non-faulty servers.

Histogram-Based Approach

In this analysis approach, the PDFs of a specific black-box metric values are determined over a window of time (of size *WinSize* seconds) at each I/O server (the current implementation uses a histogram as an approximation of the PDF, but other approaches, such as kernel density estimation techniques, are equally applicable). To compare the resulting PDFs across the different I/O servers, a standard measure, the Kullback-Leibler (KL) divergence [14], is used as the distance between two distribution functions, P and Q. The KL divergence of a distribution function, $Q()$, from the distribution function, $P()$, is given by $D(p||q) = \sum_i P(i) \log \frac{P(i)}{Qi}$. Since the KL divergence is not symmetric (i.e. $D(p||q) \neq D(q||p)$), a symmetric version of the KL divergence, given by $D'(p||q) = 0.5(D(p||q) + D(q||p))$ is used in this analysis.

The following procedure is performed for each of the metrics of interest. Note that i will be used to represent one of these metrics. Taking the PDFs of metric i for two distinct I/O servers at a time, the

pairwise KL divergences for *i* is computed. 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, within the current window, its pairwise KL-divergence for *i* with more than those of half of the other servers is anomalous. The window is shifted in time by $WinShift$ (there is thus an overlap of $WinSize - WinShift$ samples between two consecutive windows), and the analysis is repeated for the new window. A server is indicted as the culprit if it is anomalous in one or more of the chosen metrics.

Time Series-Based Approach

The previously described histogram-based approach is used for all performance metrics except `cwnd`. Unlike other metrics, `cwnd` tends to be noisy under normal conditions. This is intentional as TCP congestion control otherwise prevents synchronized connections from fully utilizing the underlying link capacity. Thus `cwnd` analysis is fundamentally different from other metrics as there is no closely-coupled peer behavior.

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. Alternatively, the distance between two log-`cwnd` values is a linear function of the number of losses that have occurred.

To support analysis of the `cwnd`, a variant of the time-series analysis algorithm describe in § 6.1 is used. First, a time-series is generated by performing a moving average on `cwnd` with a window size of 31. This attenuates the effect of sporadic transmission timeout events while still affording reasonable diagnosis latencies (i.e., under one minute). Next, the time-series is converted to a logarithmic scale by taking the common logarithm of averaged `cwnd` values. Then, every second, a representative (median) value is computed of the log-`cwnd` values. A server is flagged as anomalous if its log-`cwnd` is more than a predetermined threshold—determined by ROC-based threshold selection—below this representative value.

6.2.2 Checklist for Root-Cause Analysis

In addition to indicting the culprit node, one can also infer the resource that is the root-cause of the fault. The following checklist diagnoses the root cause when any of the steps in the decision-making sequence below turns out to be positive (answer is `YES`). If a step is positive, one halts at that step, and arrives at the root-cause and culprit node—one does not need to proceed further. If a step is negative (answer is not `YES`), one proceeds to the next step of decision-making.

- Observe peer divergence in disk throughput (`rd_sec` or `wr_sec`). If `YES`, declare disk-hog fault and indict culprit node.
- Observe peer divergence in disk latency (`await`). If `YES`, declare disk-busy fault and indict culprit node.
- Observe peer divergence in network throughput (`rxbyt` or `txbyt`). If `YES`, declare net-hog fault and indict culprit node.
- Observe peer divergence in network congestion (`cwnd`). If `YES`, declare packet-loss fault and indict culprit node.

Chapter 7

Results

7.1 Syscall-Based Diagnosis Results

Table 7.1 shows the accuracy (true- and false-positive rates[1]) of the syscall-based algorithm for diagnosing performance faults using different syscalls for the different workloads. For each fault, only those syscalls causing a non-zero true/false positive are shown.

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. The `postmark` workload is included in both halves of the table since it includes read as well as write syscalls.

The low false-positive rates are an artifact of 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.

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`	0.22	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 7.1: Results of syscall-based diagnosis. TP (FP) = true (false) positive ratio.

[1] 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.

Fault	Metric	ddr		ddw		iozoner		iozonew		postmark		Fault Overhead	
		TP	FP	TP	FP	TP	FP	TP	FP	TP	FP	Min	Max
disk-hog	rd_sec	0.9	0	0.9	0	0.9	0	0.9	0	0.9	0	13.1%	43.6%
	await	0.9	0	0.9	0	0.9	0	0.9	0	0.9	0		
	svctm	0	0	0.9	0	0	0	0.9	0	0.9	0		
disk-busy	rd_sec	0	0	0	0	0	0	0	0.1	0	0	22.8%	41.5%
	await	0.9	0	0.9	0	0.9	0	0.9	0	0.9	0		
	svctm	0.9	0	0.9	0	0.9	0	0.9	0	0.9	0		
write-network-hog	rxbyt	0.7	0	0.7	0	0.7	0	0.7	0	0.7	0	−4.1%	8.3%
	txbyt	0	0	0.9	0	0	0	0.9	0	0.9	0.1		
read-network-hog	rxbyt	0.7	0	0	0	0.8	0	0	0	0.7	0	9.4%	31.8%
	txbyt	0.9	0	0.9	0	1	0	0.9	0	0.9	0.1		
receive-pktloss	txbyt	0	0	0.2	0	0	0	0.4	0	0	0	−0.8%	18.5%
	cwnd	0	0	1	0	0	0	1	0	0	0.1		
send-pktloss	txbyt	0	0	0	0	0	0	0	0	0	0.1	−4.0%	32.4%
	cwnd	1	0	0	0	1	0	0	0	0	0		

Table 7.2: Results of PVFS performance metric-based diagnosis. TP (FP) = true (false) positive ratio.

7.2 Performance Metric-Based Diagnosis Results

7.2.1 PVFS Results

Table 7.2 shows the accuracy (true- and false-positive rates) of the performance metric-based algorithm for diagnosing performance faults for the different workloads in PVFS. For each fault, only those metrics causing a non-zero true/false positive are shown. The table also shows the minimum and maximum fault overheads, calculated as the increase in workload runtime with respect to their fault-free counterparts.

As exhibited by the Fault Overhead columns in Table 7.2, faults do not manifest equally on all workloads. Three faults, *disk-hog*, *disk-busy*, and *read-network-hog*, have significant (≈10% increase or above) impact on workload runtime for all workloads. The *receive-pktloss* and *send-pktloss* only have significant impact on workload runtime for write-heavy and read-heavy workloads respectively. Since the other workloads have minimal network throughput demands,[2] any packets lost due to this fault are quickly recovered by TCP. Thus, it is important not to count "false negatives" for these workloads against the diagnosis strategy as no performance problem is actually observed. However, it is important to count false positives for these workloads as they contribute to misdiagnosis. Finally, the *write-network-hog* fault has no significant impact on any workload,[3] so its false negatives are also to be discarded.

For faults that do have significant impact on the respective workloads, the checklist for root-cause analysis (§ 6.2.2) indicts faulty nodes with a 92.6% true-positive rate and 1.43% false-positive rate. The checklist diagnoses the root cause with a 89.5% true-positive rate and 3.14% false-positive rate. Syscall-based diagnosis, in comparison, indicts faulty nodes with a 51.0% true-positive rate and 0.57% false-positive rate. Thus, performance metric-based diagnosis provides greater confidence over syscall-based diagnosis.

[2]Opposing-flow dd/IOzone workload only transmit ACKs, and PostMark is seek-bound.

[3]Most significantly impacted workload is postmark at 8.3%, all others are < 3.5%.

Fault	Metric	ddr		ddw		iozoner		iozonew		postmark		Fault Overhead	
		TP	FP	TP	FP	TP	FP	TP	FP	TP	FP	Min	Max
None (control)	rd_sec		0.1		0		0		0		0		
disk-hog	rd_sec	0.1	0.1	0.9	0	0.1	0	0.9	0	0.9	0	6.1%	27.6%
	await	0	0	0.9	0.1	0	0	0.9	0.1	0.9	0		
	svctm	0	0	0.9	0	0	0	0.9	0	0.9	0		
disk-busy	rd_sec	0.1	0.1	0	0	0	0	0	0	0	0	19.0%	54.1%
	await	0.9	0	0.9	0.1	0.9	0	0.9	0	0.9	0		
	svctm	0.9	0	0.9	0	0.9	0	0.9	0	0.9	0		
write-network-hog	rd_sec	0	0.1	0	0	0	0	0	0	0	0	−2.5%	16.4%
	await	0	0	0.1	0.1	0	0	0	0	0	0		
	rxbyt	0.7	0	0.7	0	0.7	0	0.7	0	0.7	0		
	cwnd	0	0	0	0.1	0	0	0	0	0	0		
read-network-hog	rd_sec	0.1	0.1	0	0.1	0	0	0	0	0	0	−1.2%	34.6%
	rxbyt	0.7	0	0	0	0.8	0	0	0	0.7	0		
	txbyt	1	0	0.9	0	1	0	0.9	0	0.9	0		
receive-pktloss	rd_sec	0	0.1	0	0	0	0	0	0	0	0	9.8%	53.9%
	await	0	0	0	0.1	0	0	0	0	0	0		
	txbyt	0	0	0	0.1	0	0	0	0	0	0		
	cwnd	0	0	1	0	0	0	1	0	0	0		
send-pktloss	rd_sec	0.1	0.1	0	0	0	0	0	0	0	0	−1.0%	48.0%
	await	0	0	0	0.1	0	0	0	0	0	0		
	cwnd	1	0	0	0	1	0	0	0	0	0		

Table 7.3: Results of Lustre performance metric-based diagnosis. TP (FP) = true (false) positive ratio.

7.2.2 Lustre Results

Table 7.3 shows the accuracy (true- and false-positive rates) of the performance metric-based algorithm for diagnosing performance faults for the different workloads in Lustre. The true- and false-positive rates are computed using ROC thresholds generated from PVFS workloads, illustrating that the diagnosis training is flexible across both workloads and file systems. It is expected, though, that generating thresholds with Lustre would improve results.

The impact of faults (as illustrated by fault overheads) on Lustre is similar that of PVFS. However, since some workloads exhibit much greater runtime variance than under PVFS, the use of runtime overhead is less well suited for evaluating fault impact under Lustre. This is especially true for the `ddw` workload, which has a fault-free runtime variance of $\pm 10\%$. Thus, it is for this reason that `ddw` experiments are not included in the Fault Overhead columns in Table 7.3.

For faults that do have significant impact on the respective workloads, the checklist for root-cause analysis indicts faulty nodes with a 84.7% true-positive rate and 4.00% false-positive rate. The checklist diagnoses the root cause with a 83.2% true-positive rate and 5.14% false-positive rate. Thus, performance metric-based diagnosis for Lustre provides confidence comparable to that of PVFS.

Instrumentation	File System	Overhead for Workload				
		ddr	ddw	iozoner	iozonew	postmark
syscap	PVFS	0.2%	0.6%	−0.4%	0.7%	64.4%
strace	PVFS	−0.1%	−0.8%	−3.3%	0.0%	138.8%
/proc	PVFS	0.9%	0.0%	−0.1%	−0.8%	−0.6%
/proc	Lustre	1.0%	0.3%	0.1%	0.4%	0.2%

Table 7.4: Instrumentation overhead: Increase in run-time w.r.t. non-instrumented workload.

7.3 Overheads

Table 7.4 reports overheads for three different kinds of black-box instrumentation for PVFS and one kind for Lustre for the five workloads. Overheads are calculated as the increase in mean workload runtime (for 10 iterations) with respect to their uninstrumented counterparts.

Concerning the two syscall-based instrumentation sources (`syscap` and `strace`), since four of the 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 runtime 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).

Concerning `/proc`-based instrumentation, because it does not alter the operation of syscalls, its overheads are more modest, making it more appropriate for metadata-heavy workloads. Indeed, all runtime overheads are (< 1%) for `/proc`, thus its added runtime overhead is negligible.

Chapter 8

Conclusions & Future Work

8.1 Syscalls vs. Performance Metrics

In § 7 it is determined that OS-level performance metrics provides greater diagnosis confidence, superior root cause determination, as well as lower overheads than syscall statistics. Thus, it appears that performance metrics would be the preferred approach over syscall statistics. However, there are other advantages and disadvantages of syscall statistics compared to performance metrics that should be considered.

8.1.1 Disadvantages of Syscalls

Currently, the use of syscall statistics in performance diagnosis has limited appeal for these reasons:

- *High data and resource demands.* In the current design of `syscap` all syscall events for a process are recorded. The most syscall-conservative workload (`ddw`) generates an average of 11,000 syscall events and 1 MB of data per second per server. This is data that needs to either be parsed and processed (a cumbersome task in its own right) or at least compressed and sent across the network for real-time analysis. This problem could be mitigated by altering the design of `syscap` to include the metric extraction, and only process the subset of syscalls necessary to derive the appropriate statistics. Alternatively, *sketches* [7] may be used to sample syscall events at rates much less than the frequency of the events themselves, which allows less data and fewer resources to be used.

- *Process-local viewpoint.* `syscap` is currently only used to instrument PVFS client and server processes, restricting the vantage point of `syscap` to PVFS activity only. `/proc` instrumentation, on the other hand, has a global view of node activity and can see the behavior of both PVFS and rogue third-party processes. This allows OS-level performance metrics to diagnose more fault classes, and provide an extra dimension of root-cause analysis than syscalls can provide for performance problems.

- *Opaque caches.* I/O syscall requests that result in cache hits due to prefetching, or cache writebacks with delayed writes, record the service time of memory operations as opposed to the corresponding disk operations. In experiments Direct I/O is used to avoid this behavior. It is unclear how exactly this may skew training thresholds if Direct I/O were not used.

- *Cross-syscall manifestations.* The `dread` and `dwrite` metrics assume that syscall service time is dominated by the wait time of a disk request (sum of request queue time and disk service time). However, syscalls may be delayed by processing of soft interrupt and context switches even when disks do service requests timely. This may result in other resource exhaustion problems, which would increase the number of soft interrupts and manifest as anomalous `dread` or `dwrite` metrics. Indeed this is reflected in the results for *read-network-hog* faults that occasionally show up in the `dwrite` metrics.

8.1.2 Advantages of Syscalls

Despite the previous disadvantages, the use of syscalls and syscall statistics may still be advantages as they:

- *Can diagnose non-performance faults.* [21] describes the use of `syscap` in diagnosis of non-performance-related propagated error and server crash/hang faults in addition to performance problems. Since the diagnosis of these non-performance problems currently require the use of syscall instrumentation, syscall statistics may still be used as part of a comprehensive problem diagnosis strategy.
- *Have potential to characterize workloads.* The diagnosis strategies described in this work rely on peer-comparison to determine the performance characteristics of a normal workload, through which anomalous performance may be deduced. In traditional remote file systems (e.g., NFS, AFS, etc.) that lack peer I/O servers, additional context is required in order to determine if changes in performance are due to intended workload change or performance faults. Syscall event streams are a rich source of contextual information concerning application behavior. By observing the types of requests made by an application, one may be able to characterize workloads in a manner sufficient to determine if the resulting performance matches expectations.

8.2 Experiences

This section describes some of the experiences encountered in diagnosing performance problems in parallel file systems, highlighting counterintuitive or unobvious issues that arose during experimentation.

8.2.1 Heterogeneous Hardware

Peer-comparison diagnosis relies on peers exhibiting similar performance characteristics. Systems comprised of heterogeneous hardware, most notably hardware of different make or models, often exhibit different performance characteristics that may violate this assumption. Unfortunately, even supposedly homogeneous hardware (same make, model, stepping, etc.) may exhibit slightly different performance characteristics that impede diagnosis. Most often these differences exhibit themselves when the devices are pushed to performance extremes such as disks or network interfaces under full load.

The diagnosis techniques developed here are able to compensate for some deviations in hardware performance as long as algorithm training is performed under extreme workloads where these performance distinctions are exhibited. In this case similar performance deviations will be tolerated as normal behavior. The tradeoff, however, is that performance faults of lower severity (such that their performance impact is below the level of normal deviation) may be masked.

A secondary concern of heterogeneous hardware are 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. Thus nodes that have different memory configurations will have different caching semantics which may exhibit as non-linear changes in performance that are not easily accounted for in training. It is more important that such factors (namely, amount of system memory) be consistent across nodes.

8.2.2 Disk Saturation

Disk saturation occurs when the rate of incoming I/O requests exceeds the rate at which the disk can service them, which results in queuing of backlogged requests and response delays. The most immediate indication of disk saturation is when `%util` is at or near 100%. Secondary indications are an increase of `avgqu-sz` into the 10s or 100s range, and a significant increase in `await` as compared to `svctm` (queuing delay).

For systems or workloads where disks are not (or cannot) be saturated by normal usage, disk saturation is a good indication of disk-hog or disk-busy conditions. Unfortunately, many application and cluster configurations result in disk saturation for normal workloads. Under ideal conditions, disk saturation would occur at the same time across all nodes, and thus peer-divergence would be minimal despite value changes across many metrics. Unfortunately, due slight difference in device performance characteristics, it is most often the case that a single disk reaches saturation before others. When this occurs there is a tendency for that disk to remain in saturation while the `%util` of other disks backoff, which exactly mimics the effect of a disk-busy fault on that node despite no increase in workload (see Observation 6 in § 4.2 for an explanation of the effect). Thus disk-busy faults must be distinguished from normal saturation by the degree of the effect (deviation in `await` or `%util`).

8.2.3 Buffer Cache

During experimentation it was discovered that the behavior of the Linux buffer cache confounds disk metrics for PVFS write workloads. The modal behavior of the buffer cache greatly influences disk operation despite only minor differences in node state and nearly identical client requests. [33] describes the behavior of the Linux buffer cache which is simplified here to a model with three modes: (i) periodic write, (ii) background write, and (iii) foreground write.

In periodic write mode, write requests are committed to the Linux buffer cache as dirty pages which remain in the cache until they expire, typically after 30 seconds. A `pdflush` thread wake up periodically, typically every 5 seconds, to write pages that have recently expired. Thus, disk throughput consists of a periodic aggregated write profile.

In background write mode, write requests are received with sufficient rate such that a threshold quantity (`dirty_background_ratio`—typically 10%) of active memory fills with dirty pages before they expire. Once this threshold is reached, a `pdflush` thread writes a sufficient quantity of unexpired pages to reduce the number of dirty pages below this threshold again. For a steady flow of incoming write requests, this results in a steady state background write behavior where pages are written to disk with the same average throughput as incoming requests.

In foreground write mode, write requests are received at a rate that exceeds that which `pdflush` can

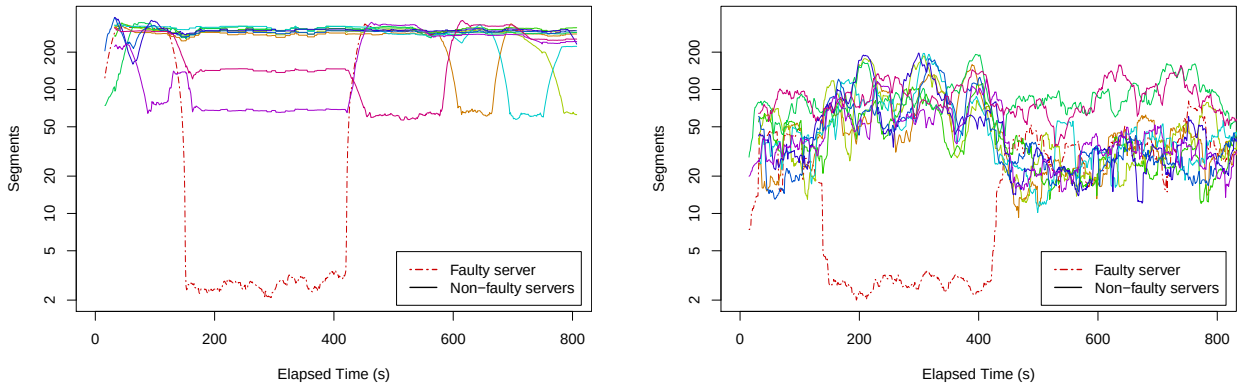

Figure 8.1: Single (left) and multiple (right) client `cwnd`s for `ddw` workloads with *receive-pktloss* faults.

write out in background mode. Once the quantity of surplus pages in the buffer cache exceeds `dirty_ratio` (typically 40%) of active memory, Linux then forces user processes to commit dirty pages directly to disk, blocking any process that attempts to write to the disk until this is done.

Disk performance metrics tends to be similar across nodes as long as all nodes are within periodic or background write modes. However, if nodes are operating in different modes, metrics may deviate significantly. This is particularly so if one or more of the nodes is in foreground write mode and the corresponding disks are saturated. Even if all nodes are performing foreground writes, the disk performance metrics may still vary significantly depending on the contents of the cache and the number of queued requests, both of which are dependent on the time at which the foreground write threshold is reached, which itself is a function of node memory and underlying device performance.

The buffer cache has also been witnessed to negatively influence recovery time due to lingering manifestations even after a fault has been removed. In the event of a disk-hog or disk-busy fault that results in disk saturation on the faulty node and a buildup of dirty pages, once the fault is removed it may take time to flush the expired dirty pages and this extra recovery workload may carry seek penalties that delays write performance as compared to fault-free nodes.

For these reasons the experiments in this work avoid the use of the buffer cache when monitoring performance of PVFS. Fortunately, PVFS 2.8.0 includes a Direct I/O storage method that bypasses the buffer cache while providing improved performance for high-bandwidth writes. In contrast, Lustre, which uses both client and server side caching, has not exhibited the same disk metric deviations across I/O servers, and thus it was not sought to modify its caching properties for this work.

8.2.4 Multiple Clients

Performance metrics for single client workloads vs. multiple client workloads differ in some significant ways. For example, in PVFS clusters with caching enabled, the buffer cache aggregates contiguous small writes for single client workloads which considerably improves throughput performance. The buffer cache does not aggregate as well small writes from multiple client workloads and the penalty due to interfering seeks greatly reduces throughput and pushes disks towards saturation. Thus, for PVFS large application write record sizes (preferably order MB per server) are needed to maximize multiple client performance.

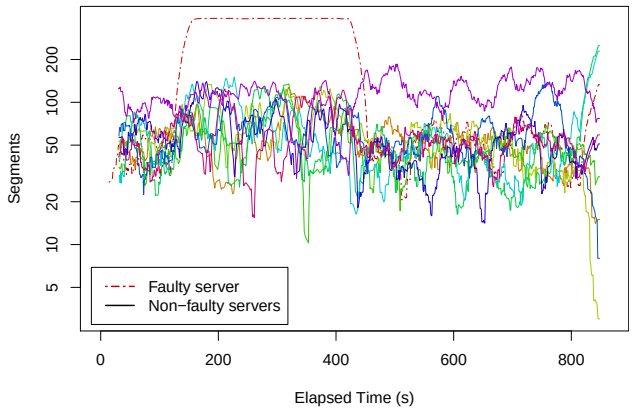

Figure 8.2: Disk-busy fault influence on faulty server's `cwnd` for `ddr` workload.

Single vs. multiple clients also has an impact on network congestion (see Figure 8.1). Single client write workloads result in bulk data transfers being sourced from a single point with relatively little network congestion (opposing flows comprise of only server ACKs). This results in steady client `cwnd` figures that deviate sharply during fault. Multiple client write workloads result in bulk data transfers source from many points that interfere with each other and the resulting congestion creates chaotic, widely varying `cwnd` figures. While faulty `cwnds` are still distinguishable, it highlights the need to train on worst case workloads.

8.2.5 Cross-Resource Fault Influences

It is given that faults will have cross-metric influence on a single resource. For example, a disk-hog fault immediately results in increased throughput on the faulty disk, but since it also pushes that disk into saturation it results in increased request queuing and latency. However, there are instances in which faults affecting one resource may manifest in metrics concerning another resource in an unintuitive way.

Consider a disk-busy fault's influence on the faulty server's sending congestion windows during a large read workload (see Figure 8.2). `cwnd` values are updated only when a server is both sending and experiencing congestion, thus the `cwnd` metric, perhaps counterintuitively, does not describe the amount of present network congestion when a server is *not* sending data. In the case of the disk-busy fault, data transfer occurs between a single client and the servers in three distinct phases: in (i) the clients send requests to each of the servers, in (ii) the fault-free servers respond quickly then fall idle, and in (iii) the faulty server responds after a significantly delayed disk read request.

In PVFS, the lack of client readahead blocks clients on the faulty server responses, effectively synchronizing the clients so that the above phases occur simultaneously on all clients. Bulk data transfers occur in phases ii and iii, but during phase ii all fault-free servers are transmitting which creates network congestion and chaotic `cwnd` values whereas during phase iii only the faulty server is transmitting, and in the inverse of the single client scenario, it experiences virtually no congestion and thus maintains a stable, high `cwnd` value. This creates a divergence in the faulty server's `cwnd` as compared to its peers, indicating a potential network-related fault when there is only a disk-busy fault present.

This cross-resource fault influence may question the validity of the root-cause analysis checklist described in § 6.2.2. However, since the TCP congestion-control metrics are the only ones known to suffer

from this counterintuitive property, it may be mitigated by placing greater confidence in disk metric anomalies over network metrics in the checklist. Note that since Lustre does use client readahead, read calls are not as well phase synchronized across clients, and thus, this influence does not manifest as severely.

8.3 Future Work

An in-kernel utility for syscall instrumentation is under consideration for development. Such a utility would record syscalls and their arguments to an internal buffer and provide interfaces for reading the buffer and for specifying which syscalls should be logged. By performing selective logging of syscalls and avoiding additional context switches, this utility would significantly improve worst-case instrumentation overheads and decrease the data/resource burden, which also allows for faster analysis.

An incremental improvement on this work is to improve metric analysis by using a different measure for comparing PDFs. KL divergence, since it performs bin-by-bin comparison, tends to be overly sensitive to changes in the WinSize and WinShift parameters. This results in poor true- and false-positive rates when these parameters are not properly tuned (i.e., via ROC curves) for the specific file system cluster. It is speculated that other measures, such as the Earth Mover's Distance (EMD) [40] would enable a more robust analysis that is less sensitive to tuning, enabling easier deployment. OSprof [20] uses EMD on similar data with good results.

Next, the scope of problem diagnosis will be expanded to include additional file systems (for which preliminary work has already been performed on GlusterFS and Ceph), realistic workloads, and additional faults. To support the latter two goals, work is underway to start collecting performance metrics and other workload data from production/stable PVFS deployments. This workload data will be used to target more representative workloads.

Finally, it is desired to seek a new instrumentation source in the messaging (e.g., PVFS BMI, Lustre RPC) layers of parallel file systems. By observing client-server I/O requests on the wire, one could obtain more accurate workload and performance characteristics than those provided at the syscall or OS levels. Such instrumentation has not been previously sought as the proprietary nature of file system protocols has meant that any implementation is applicable to only a single, existing file system. However, with the recent standardization of Parallel NFS (pNFS) [32] as a common interface for parallel storage, instrumentation of pNFS in addition to proprietary protocols would allow both current and future file systems to benefit from problem diagnosis at the messaging layer.

8.4 Conclusion

This work demonstrates and compares two black-box analysis strategies for performance faults in parallel file systems. In doing so, a number of (empirically-based) observations are made about PVFS's and Lustre's behavior with regard to performance faults, and these observations are used in the development of the diagnosis algorithms. Both diagnosis strategies are shown to detect and diagnose a number of performance problems with a low false-positive rate. Finally, a number of experimental experiences are reported, which contribute to understanding how to improve and better approach problem diagnosis in future work.

Bibliography

[1] M. K. Agarwal, M. Gupta, V. Mann, N. Sachindran, N. Anerousis, and L. Mummert. Problem determination in enterprise middleware systems using change point correlation of time series data. In *Proceedings of 10th IEEE/IFIP Network Operations and Management Symposium (NOMS '06)*, pages 471–482, Vancouver, BC, Apr. 2006.

[2] 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 (SOSP '03)*, pages 74–89, Bolton Landing, NY, Oct. 2003.

[3] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems*, 19(4):483–518, Nov. 2001.

[4] A. Aranya, C. P. Wright, and E. Zadok. Tracefs: A file system to trace them all. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST '04)*, pages 129–145, San Francisco, CA, Apr. 2004.

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

[6] 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 (OSDI '04)*, pages 259–272, San Francisco, CA, Dec. 2004.

[7] S. Bhatia, A. Kumar, M. E. Fiuczynski, and L. Peterson. Lightweight, high-resolution monitoring for troubleshooting production systems. In *Proceedings of the 8th Symposium on Operating Systems Design and Implementation (OSDI '08)*, pages 103–116, San Diego, CA, Dec. 2008.

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

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

[10] 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*, pages 317–327, Atlanta, GA, Oct. 2000.

[11] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC '04)*, pages 36–43, New York, NY, May 2004.

[12] 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 (DSN '02)*, pages 595–604, Bethesda, MD, June 2002.

[13] 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 (SOSP '05)*, pages 105–118, Brighton, UK, Oct. 2005.

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

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

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

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

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

[19] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky. Are disks the dominant contributor for storage failures? a comprehensive study of storage subsystem failure characteristics. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08)*, pages 111–125, San Jose, CA, Feb. 2008.

[20] N. Joukov, A. Traeger, R. Iyer, C. P. Wright, and E. Zadok. Operating system profiling via latency analysis. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*, pages 89–102, Seattle, WA, Nov. 2006.

[21] 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 (HotDep '09)*, Lisbon, Portugal, June 2009.

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

[23] 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.

[24] M. P. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger. Modeling the relative fitness of storage. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '07)*, pages 37–48, San Diego, CA, June 2007.

[25] 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*, 28(11):37–46, Nov. 2005.

[26] A. V. Mirgorodskiy. *Automated Problem Diagnosis in Distributed Systems*. PhD thesis, University of Wisconsin-Madison, Madison, WI, 2006.

[27] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Model-based failure analysis of journaling file systems. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN '05)*, pages 802–811, Yokohama, Japan, July 2005.

[28] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. IRON file systems. In *Proceedings of the 20th ACM Symposium on Operating SystemsPrinciples (SOSP '05)*, pages 206–220, Brighton, UK, Oct. 2005.

[29] 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 (NSDI '06)*, pages 115–128, San Jose, CA, May 2006.

[30] K. Shen, C. Stewart, C. Li, and X. Li. Reference-driven performance anomaly identification. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '09)*, Seattle, WA, June 2009.

[31] S. S. Shende and A. D. Malony. The Tau parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, May 2006.

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

[33] G. Smith. The linux page cache and pdflush: Theory of operation and tuning for write-heavy loads, Aug. 2007. `http://www.westnet.com/~gsmith/content/linux-pdflush.htm`.

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

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

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

[37] 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 *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '06)*, pages 3–14, Saint-Malo, France, June 2006.

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

[39] 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 (OSDI '06)*, pages 307–320, Seattle, WA, Nov. 2006.

[40] Wikipedia contributors. Earth mover's distance. *Wikipedia, The Free Encyclopedia.*, Apr. 2009. `http://en.wikipedia.org/w/index.php?title=Earth_mover%27s_distance&oldid=286004899`.