Making Problem Diagnosis Work for Large-Scale, Production Storage Systems

Michael P. Kasick and Priya Narasimhan, ***Carnegie Mellon University;***
Kevin Harms, ***Argonne National Laboratory***

https://www.usenix.org/conference/lisa13/technical-sessions/papers/kasick

This paper is included in the Proceedings of the 27th Large Installation System Administration Conference (LISA '13).

November 3–8, 2013 • Washington, D.C., USA

ISBN 978-1-931971-05-8

Making Problem Diagnosis Work for Large-Scale, Production Storage Systems

Michael P. Kasick, Priya Narasimhan
Electrical & Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213–3890
{mkasick, priyan}@andrew.cmu.edu

Kevin Harms
Argonne Leadership Computing Facility
Argonne National Laboratory
Argonne, IL 60439
harms@alcf.anl.gov

Abstract

Intrepid has a very-large, production GPFS storage system consisting of 128 file servers, 32 storage controllers, 1152 disk arrays, and 11,520 total disks. In such a large system, performance problems are both inevitable and difficult to troubleshoot. We present our experiences, of taking an automated problem diagnosis approach from proof-of-concept on a 12-server test-bench parallel-file-system cluster, and making it work on Intrepid's storage system. We also present a 15-month case study, of problems observed from the analysis of 624 GB of Intrepid's instrumentation data, in which we diagnose a variety of performance-related storage-system problems, in a matter of hours, as compared to the days or longer with manual approaches.

Tags: problem diagnosis, storage systems, infrastructure, case study.

1 Introduction

Identifying and diagnosing problems, especially performance problems, is a difficult task in large-scale storage systems. These systems are comprised of many components: tens of storage controllers, hundreds of file servers, thousands of disk arrays, and tens-of-thousands of disks. Within high-performance computing (HPC), storage often makes use of parallel file systems, which are designed to utilize and exploit parallelism across all of these components to provide very high-bandwidth concurrent I/O.

An interesting class of problems in these systems is hardware component faults. Due to redundancy, generally component faults and failures manifest in degraded performance. Due to careful balancing of the number of components and their connections, the degraded performance of even a single hardware component may be observed throughout an entire parallel file system, which makes problem localization difficult.

At present, storage system problems are observed and diagnosed through independent monitoring agents that exist within the individual components of a system, e.g., disks (via S.M.A.R.T. [11]), storage controllers, and file servers. However, because these agents act independently, there is a lack of understanding how a specific problem affects overall performance, and thus it is unclear whether a corrective action is immediately necessary. Where the underlying problem is the misconfiguration of a specific component, an independent monitoring agent may not even be aware that a problem exists.

Over the past few years, we have been exploring the use of peer-comparison techniques to identify, locate, and diagnose performance problems in parallel file systems. By understanding how individual components may exhibit differences (asymmetries) in their performance relative to peers, and, based on the presence of these asymmetries, we have been able to identify the specific components responsible for overall degradation in system performance.

Our previous work. As described in [17], we automatically diagnosed performance problems in parallel file systems (in PVFS and Lustre) by analyzing black-box, OS-level performance metrics on every file server. We demonstrated a proof-of-concept implementation of our peer-comparison algorithm by injecting problems during runs of synthetic workloads (dd, IOzone, or PostMark) on a controlled, laboratory test-bench storage cluster of up to 12 file servers. While this prototype demonstrated that peer comparison is a good foundation for diagnosing problems in parallel file systems, it did not attempt to tackle the practical challenges of diagnosis in large-scale, real-world production systems.

Contributions. In this paper, we seek to adapt our previous approach for the primary high-speed storage system of Intrepid, a 40-rack Blue Gene/P supercomputer at Argonne National Laboratory [21], shown in Figure 1. In doing so, we tackle the practical issues in making problem diagnosis work in large-scale environment, and we also evaluate our approach through a 15-month case study of practical problems that we observe and identify within Intrepid's storage system.

The contributions of this paper are:

Figure 1: Intrepid, consists of 40 Blue Gene/P racks [2].

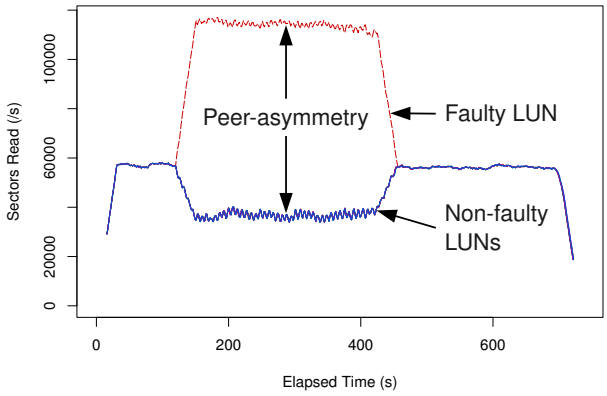

Figure 2: Asymmetry in throughput for an injected fault; provides intuition behind the peer-comparison approach that serves as a good foundation for our diagnosis [17].

- Outlining the pragmatic challenges of making problem diagnosis work in large-scale storage systems.
- Adapting our proof-of-concept diagnosis approach, from its initial target of a 12-server experimental cluster, to a 9,000-component, production environment consisting of file servers, storage controllers, disk arrays, attachments, etc.
- Evaluating a case study of problems observed in Intrepid's storage system, including those that were previously unknown to system operators.

We organize the rest of this paper as follows. We start with a description of our approach, as it was originally conceived, to work in a small-scale laboratory environment (see § 2). We then discuss the challenges of taking the initial algorithm from its origin in a limited, controllable test-bench environment, and making it effective in a noisy, 9,000-component production system (see § 3). Finally, we present the new version of our algorithm that works in this environment, and evaluate its capability to diagnose real-world problems in Intrepid's storage system (see § 4).

2 In the Beginning . . .

The defining property of parallel file systems is that they parallelize accesses to even a single file, by striping its data across many, if not all, file servers and logical storage units (LUNs) within a storage system. By striping data, parallel file systems maintain similar I/O loads across system components (peers) for all non-pathological client workloads. In our previous work [17], we hypothesized that the statistical trend of I/O loads, as reflected in OS-level performance metrics, should (i) exhibit symmetry across fault-free components, and (ii) exhibit asymmetries across faulty components. Figure 2 illustrates the intuition behind our hypothesis; the injection of a rogue workload on a spindle shared with a PVFS LUN results in a throughput asymmetry between the faulty and fault-free LUNs, where previously throughput was similar across them.

In the context of our diagnosis approach, *peers* represent components of the same type or functionality that are expected to exhibit similar request patterns. By capturing performance metrics at each peer, and comparing these metrics across peers to locate asymmetries (*peer-comparison*), we expect to be able to identify and localize faults to the culprit peer(s).

To validate our hypothesis, we explored a peer-comparison-based approach to automatically diagnose performance problems through a set of experiments on controlled PVFS and Lustre test-bench clusters [17]. In summary, these experiments are characterized by:

- Black-box instrumentation consisting of samples of OS-level storage and network performance metrics.
- Two PVFS and two Lustre test-bench clusters, containing 10 clients and 10 file servers, or 6 clients and 12 file servers, for four clusters in total.
- File servers each with a single, locally-attached storage disk.
- Experiments, both fault-free and fault-injected, approximately 600 seconds in duration, where each client runs the same file system benchmark (dd, IOzone, or PostMark) as a synthetic workload.
- Fault-injection of approximately 300 seconds in duration, consisting of two storage-related, and two network-related performance problems.
- A peer-comparison diagnosis algorithm that is able to locate the faulty server (for fault-injection experiments), and determine which of the four injected problems is present.

In [17], we evaluate the accuracy of our diagnosis with true- and false-positive rates for diagnosing the correct faulty server and fault type (if an injected fault exists). Although our initial diagnosis algorithm exhibited weaknesses that contributed to misdiagnoses in our re-

sults, overall our test-bench experiments demonstrated that peer-comparison is a viable method for performing problem diagnosis, with low instrumentation overhead, in parallel file systems.

3 Taking it to the Field

Following the promising success of our PVFS and Lustre test-bench experiments, we sought to validate our diagnosis approach on Intrepid's primary high-speed GPFS file system (we describe GPFS in § 3.2 and Intrepid's architecture in § 3.3).

In doing so, we identified a set of *new challenges* that our diagnosis approach would have to handle:

1. A large-scale, multi-tier storage system where problems can manifest on file servers, storage attachments, storage controllers, and individual LUNs.
2. Heterogeneous workloads of unknown behavior and unplanned hardware-component faults, both of which are outside of our control, that we observe and characterize as they happen.
3. The presence of system upgrades, e.g., addition of storage units that see proportionally higher loads (non-peer behavior) as the system seeks to balance resource utilization.
4. The need for continuous, 24/7 instrumentation and analysis.
5. Redundant links and components, which also exhibit changes in load (as compared to peers) when faults are present, even though the components themselves are operating appropriately.
6. The presence of occasional, transient performance asymmetries that are not conclusively attributable to any underlying problem or misbehavior.

3.1 Addressing these New Challenges

While problem diagnosis in Intrepid's storage system is based on the same fundamental peer-comparison process we developed during our test-bench experiments, these new challenges still require us to adapt our approach at every level: by expanding the system model, revisiting our instrumentation, and improving our diagnosis algorithm. Here we map our list of challenges to the subsequent sections of the paper where we address them.

Challenge #2. Tolerating heterogeneous workloads and unplanned faults are inherent features of our peer-comparison approach to problem diagnosis. We assume that client workloads exhibit similar request patterns across all storage components, which is a feature provided by parallel file system data striping for all but pathological cases. We also assume that at least half of the storage components (within a peer group) exhibit fault-free behavior. As long as these assumptions hold, our peer-comparison approach can already distinguish problems from legitimate workloads.

Challenges #1, #3, and #5. Unlike our test-bench, which consisted of a single storage component type (PVFS or Lustre file server with a local storage disk), Intrepid's storage system consists of multiple component types (file servers, storage controllers, disk arrays, attachments, etc.), that may be amended or upgraded over time, and that serve in redundant capacities. Thus, we are required to adapt our system model to tolerate each of these features. Since we collect instrumentation data on file servers (see § 4.1), we use LUN-server attachments as our fundamental component for analysis. With knowledge of GPFS's prioritization of attachments for shared storage (see § 3.3.2), we handle redundant components (challenge #5) by separating attachments into different priority groups that are separately analyzed. We handle upgrades (challenge #3) similarly, separating components into different sets based on the time at which they're added to the system, and perform diagnosis separately within each upgrade set (see § 3.3.1). Furthermore, by knowing which attachments are affected at the same time, along with the storage system topology (see § 3.3), we can infer the most likely tier and component affected by a problem (challenge #1).

Challenge #4. As in [17] we use `sadc` to collect performance metrics (see § 4.1). To make our use of `sadc` amenable to continuous instrumentation, we also use a custom daemon, `cycle`, to rotate `sadc`'s activity files once a day (see § 4.1.1). This enables us to perform analysis on the previous day's activity files while `sadc` generates new files for the next day.

Challenge #6. Transient performance asymmetries are far more common during the continuous operation of large-scale storage systems, as compared to our short test-bench experiments. Treatment of these transient asymmetries requires altering the focus of our analysis efforts and enhancing our diagnosis algorithm to use persistence ordering (see § 4.3).

3.2 Background: GPFS Clusters

The General Parallel File System (GPFS) [27] is a cluster and parallel file system used for both high-performance computing and network storage applications. A GPFS storage cluster consists of multiple file servers that are accessed by one or more client nodes, as illustrated for Intrepid in Figure 3. For large I/O operations, clients issue simultaneous requests across a local area network (e.g., Ethernet, Myrinet, etc.) to each file server. To facilitate storage, file servers may store data on local (e.g., SATA) disks, however, in most clusters I/O requests are further forwarded to dedicated storage controllers, either via direct attachments (e.g., Fibre Channel, InfiniBand) or over a storage area network.

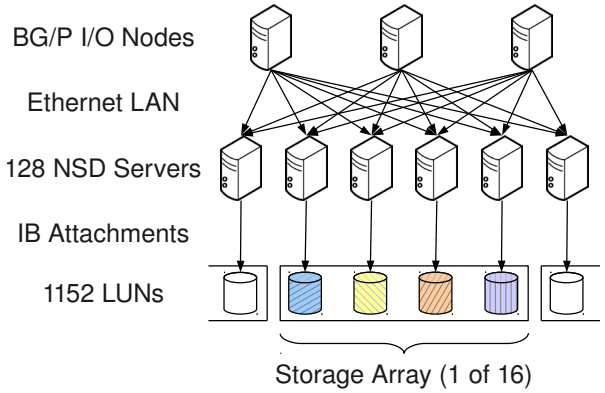

Figure 3: Intrepid's storage system architecture.

Storage controllers expose block-addressable logical storage units (LUNs) to file servers and store file system content. As shown in Figure 4, each LUN consists of a redundant disk array. Controllers expose different subsets of LUNs to each of its attached the file servers. Usually LUNs are mapped so each LUN primarily serves I/O for one (primary) file server, while also allowing redundant access from other (secondary) file servers. This enables LUNs to remain accessible to clients in the event that a small-number of file servers go offline. Controllers themselves may also be redundant (e.g., coupled "A" and "B" controllers) so that LUNs remain accessible to secondary file servers in the event of a controller failure.

The defining property of parallel file systems, including GPFS, is that they parallelize accesses to even a single file, by striping its data across many (and in a common configuration, across all) file servers and LUNs. For example, when performing large, sequential I/O, clients may issue requests, corresponding to adjacent stripe segments, round-robin to each LUN in the cluster. LUNs are mapped to file servers so that these requests are striped to each file server, parallelizing access across the LAN, and further striped across the primary LUNs attached to file servers, parallelizing access across storage attachments.

The parallelization introduced by the file system, even for sequential writes to a single file, ensures that non-pessimistic workloads exhibit equal loads across the cluster, which in turn, should be met with balanced performance. Effectively, just as with other parallel file systems, GPFS exhibits the characteristics that make peer-comparison a viable approach for problem diagnosis. Thus, when "hot spots" and performance imbalances arise in a cluster, we hypothesize them to be indicative of a performance problem. Furthermore, by instrumenting each file server in the cluster, we can observe the performance of file servers, storage controllers, and LUNs, from multiple perspectives, which enables us to localize problems to the components of the cluster where performance imbalance is most significant.

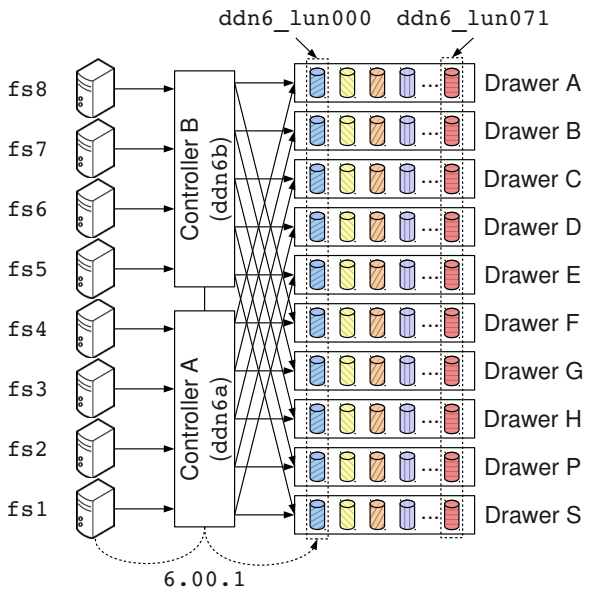

Figure 4: Storage array subarchitecture, e.g., `ddn6`.

3.3 Intrepid's Storage System

The target of our case study is Intrepid's primary storage system, a GPFS file system that consists of 128 Network Shared Disk (NSD) servers (`fs1` through `fs128`) and 16 DataDirect Networks S2A9900 storage arrays, each with two storage controllers [21]. As illustrated in Figure 4, each storage array exports 72 LUNs (`ddn6_lun000` through `ddn21_lun071`) for Intrepid's GPFS file system, yielding a 4.5 PB file system comprised from 1152 LUNs and 11,520 total disks. At this size, this storage system demands a diagnosis approach with scalable data volume and an algorithm efficient enough to perform analysis in real-time with modest hardware. In addition, because our focus is on techniques that are amenable to such production environments, we require an approach with a low instrumentation overhead.

3.3.1 System Expansion

Of the 72 LUNs exported by each storage array, 48 were part of the original storage system deployment, while the other 24 were added concurrently with the start of our instrumentation to expand the system's capacity. Since the 24 LUNs added in each storage array (384 LUNs total) were initially empty, they observe fewer reads and more writes, and thus, exhibit non-peer behavior compared to the original 48 LUNs in each array (768 LUNs total). As our peer-comparison diagnosis approach performs best on LUNs with similar workloads, we partition Intrepid into "old" and "new" LUN sets, consisting of 768 and 384 LUNs respectively, and perform our diagnosis separately within each set.

3.3.2 Shared Storage

Each Intrepid LUN is redundantly attached to eight GPFS file servers with a prioritized server ordering defined in a system-wide configuration. We denote these LUN-server attachments with the convention `controller.lun.server`, e.g., `6.00.1` or `21.71.128`.

GPFS clients, when accessing a LUN, will route all I/O requests through the highest-priority, presently-available server defined for that LUN. Thus, when all servers are online, client I/O requests route through the primary server defined for a given LUN. If the primary server is unavailable, requests route through the LUN's secondary, tertiary, etc., servers based on those servers availability.

Since redundant attachments do not have equal priority for a given LUN, this effectively creates eight system-wide priority groups consisting of equal-priority LUN-server attachments, i.e., the first priority-group consists of all primary LUN-server attachments, the second priority-group consists of all the secondary LUN-server attachments, etc. Combined with the "system expansion" division, the total of 9216 LUN-server attachments (1152 LUNs $\times$ 8 redundantly attached servers) must be analyzed in 16 different peer groups (8 priority groups $\times$ 2 for "old" vs. "new" LUNs) in total.

4 Making it Work for Intrepid

As we apply our problem-diagnosis approach to large storage systems like Intrepid's, our primary objective is to locate the most problematic LUNs (specifically LUN-server attachments that we refer to as "LUNs" henceforth) in the storage system, which in turn, reflect the location of faults with greatest performance impact. This process consists of three stages:

Instrumentation, where we collect performance metrics for every LUN (see § 4.1);

Anomaly Detection, where we identify LUNs that exhibit anomalous behavior for a specific window of time (see § 4.2);

Persistence Ordering, where we locate the most problematic components by their persistent impact on overall performance (see § 4.3).

4.1 Instrumentation

For our problem diagnosis, we gather and analyze OS-level storage performance metrics, without requiring any modifications to the file system, the applications or the OS. As we are principally concerned with problems that manifest at the layer of NSD Servers and below (see Figure 3), the metrics that we gather and utilize consist of the storage-metric subset of those collected in our previous work [17].

Metric	Significance
`tps`	Number of I/O (read and write) requests made to (a specific) LUN per second.
`rd_sec`	Number of sectors read from the LUN per second.
`wr_sec`	Number of sectors written to the LUN per second.
`avgrq-sz`	Average size (in sectors) of the LUN's I/O requests.
`avgqu-sz`	Average number of the LUN's queued I/O requests.
`await`	Average time (in milliseconds) that a request waits to complete on the LUN; includes queuing delay and service time.
`svctm`	Average (LUN) service time (in milliseconds) of I/O requests; does not include any queuing delay.
`%util`	Percentage of CPU time in which I/O requests are made to the LUN.

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

In Linux, OS-level performance metrics are made available as text files in the `/proc` pseudo file system. Table 1 describes the specific metrics that we collect. We use sysstat's `sadc` program [15] to periodically gather storage and network performance metrics at a sampling interval of one second, and record them in activity files. For storage resources, sysstat provides us with the throughput (`tps`, `rd_sec`, `wr_sec`) and latency (`await`, `svctm`) of the file server's I/O requests to each of the LUNs the file server is attached to. Since our instrumentation is deployed on file servers, we actually observe the compound effect of disk arrays, controllers, attachments, and the file server on the performance of these I/O requests.

In general we find that `await` is the best single metric for problem diagnosis in parallel file systems as it reflects differences in latency due to both (i) component-level delays (e.g., read errors) and (ii) disparities in request queue length, i.e., differences in workload. Since workload disparities also manifest in changes in throughput, instances in which `await` is anomalous but not `rd_sec` and `wr_sec` indicate a component-level problem.

4.1.1 Continuous Instrumentation

As our test-bench experiments in [17] were of relatively short duration ($\sim$600 s), we were able to spawn instances of `sadc` to record activity files for the duration of our experiments, and perform all analysis once the experiments had finished and the activity files were completely written. For Intrepid, we must continuously instrument and collect data, while also periodically performing offline analysis. To do so, we use a custom daemon, `cycle`, to spawn daily instances of `sadc` shortly after midnight UTC, at which time we are able to collect the previous day's activity files for analysis.

Although the `cycle` daemon performs a conceptually simple task, we have observed a number of practical issues in deployment that motivated the development of robust time-management features. We elaborate on our

experiences with these issues in § 6. To summarize, the present version of `cycle` implements the following features:

- Records activity files with filenames specified with an ISO 8601-formatted UTC timestamp of `sadc`'s start time.
- Creates new daily activity files at 00:00:05 UTC, which allows up to five seconds of clock backwards-correction without creating a second activity file at 23:59 UTC on the previous day.
- Calls `sadc` to record activity files with a number of records determined by the amount of time remaining before 00:00:05 UTC the next day, as opposed to specifying a fixed number of records. This prevents drifts in file-creation time due to accumulating clock corrections. It also allows for the creation of shorter-duration activity files should a machine be rebooted in the middle of the day.

4.2 Anomaly Detection

The purpose of anomaly detection is to determine which storage LUNs are *instantaneously* reflecting anomalous, non-peer behavior. To do so, we use an improved version of the histogram-based approach described in [17].

Inevitably, any diagnosis algorithm has configurable parameters that are based on the characteristics of the data set for analysis, the pragmatic resource constraints, the specific analytical technique being used, and the desired diagnostic accuracy. In the process of explaining our algorithms below, we also explain the intuition behind the settings of some of these parameters.

Overview. To find the faulty component, we peer-compare storage performance metrics across LUNs to determine those with anomalous behavior. We analyze one metric at a time across all LUNs. For each LUN we first perform a moving average on its metric values. We then generate the Cumulative Distribution Function (CDF) of the smoothed values over a time window of *WinSize* samples. We then compute the distance between CDFs for each pair of LUNs, which represents the degree to which LUNs behave differently. We then flag a LUN as anomalous over a window if more than half of its pairwise CDF distances exceed a predefined threshold. We then shift the window by $WinShift$ samples, leaving an overlap of $WinSize - WinShift$ samples between consecutive windows, and repeat the analysis. We classify a LUN to be faulty if it exhibits anomalous behavior for at least k of the past $2k-1$ windows.

Downsampling. As Intrepid occasionally exhibits a light workload with requests often separated by periods of inactivity, we downsample each storage metric to an interval of 15 s while keeping all other diagnosis parameters the same. This ensures that we incorporate a reasonable quantity of non-zero metric samples in each comparison window to detect asymmetries. It also serves as a scalability improvement by decreasing analysis time, and decreasing storage and (especially) memory requirements. This is a pragmatic consideration, given that the amount of memory required would be otherwise prohibitively large[1]

Since `sadc` records each of our storage metrics as a rate or time average, proper downsampling requires that we compute the metric's cumulative sum, that we then sample (at a different rate), to generate a new average time series. This ensures any work performed between samples is reflected in the downsampled metric just as it is in the original metric. In contrast, sampling the metric directly would lose any such work, which leads to inaccurate peer-comparison. The result of the downsampling operation is equivalent to running `sadc` with a larger sampling interval.

Moving Average Filter. Sampled storage metrics, particularly for heavy workloads, can contain a large amount of high-frequency (relative to sample rate) noise from which it is difficult to observe subtle, but sustained fault manifestations. Thus, we employ a moving average filter with a 15-sample width to remove this noise. As we do not expect faults to manifest in a periodic manner with a periodicity less than 15 samples, this filter should not unintentionally mask fault manifestations.

CDF Distances. We use cumulative histograms to approximate the CDF of a LUN's smoothed metric values. In generating the histograms we use a modified version of the Freedman-Diaconis rule [12] to select the bin size, $BinSize = 2\,\mathrm{IQR}(x)\,WinSize^{-1/3}$, and number of bins, $Bins = \lceil \mathrm{Range}(x)/BinSize \rceil$ where x contains samples across *all* LUNs in the time window. Even though the generated histograms contain samples from a single LUN, we compute *BinSize* using samples from all LUNs to ensure that the resulting histograms have compatible bin parameters and, thus, are comparable. Since each histogram contains only *WinSize* samples, we compute *BinSize* using *WinSize* number of observations. Once histograms are generated for each LUN's values, we compute for each pair of histograms P and Q the (symmetric) distance: $d(P,Q) = \sum_{i=0}^{Bins} |P(i) - Q(i)|$, a scalar value that represents how different two histograms, and thus LUNs, are from each other.

Windowing and Anomaly Filtering. Looking at our test-bench experiments [17], we found that a *WinSize* of ∼60 samples encompassed enough data such that our components were observable as peers, while also

[1] We frequently ran out of memory when attempting to analyze the data of a single metric, sampling at 1 s, on machines with 4 GB RAM.

maintaining a reasonable diagnosis latency. We use a $WinShift$ of 30 samples between each window to ensure a sufficient window overlap (also 30 samples) so as to provide continuity of behavior from an analysis standpoint. We classify a LUN as faulty if it shows anomalous behavior for 3 out of the past 5 windows ($k = 3$). This filtering process reduces many of the spurious anomalies associated with sporadic asymmetry events where no underlying fault is actually present, but adds to the diagnosis latency. The $WinSize$, $WinShift$, and k values that we use, along with our moving-average filter width, were derived from our test-bench experiments as having providing the best empirical accuracy rates and are similar to the values we published in [17], while also providing for analysis windows that are round to the half-minute. The combined effects of downsampling, windowing, and anomaly filtering result in a diagnosis latency (the time from initial incident to diagnosis) of 22.5 minutes.

4.2.1 Threshold Selection

The CDF distance thresholds used to differentiate faulty from fault-free LUNs are determined through a fault-free training phase that captures the maximum expected deviation in LUN behavior. We use an entire day's worth of data to train thresholds for Intrepid. This is not necessarily the minimum amount of data needed for training, but it is convenient for us to use since our experiment data is grouped by days. We train using the data from the first (manually observed) fault-free day when the system sees reasonable utilization. If possible, we recommend training during stress tests that consist of known workloads, which are typically performed before a new or upgraded storage system goes into production. We can (and do) use the same thresholds in on-going diagnosis, although retraining would be necessary in the event of a system reconfiguration, e.g., if new LUNs are added. Alternatively we could retrain on a periodic (e.g., monthly) basis as a means to tolerate long-term changes to LUN performance. However, in practice, we have not witnessed a significant increase in spurious anomalies during our 15-month study.

To manually verify that a particular day is reasonably fault-free and suitable for training, we generate, for each peer group, plots of superimposed `await`s for all LUNs within that peer group. We then inspect these plots to ensure that there is no concerning asymmetry among peers, a process that is eased by the fact that most problems manifest as observable loads in normally zero-valued non-primary peer groups. Even if training data is not perfectly fault-free (either due to minor problems that are difficult to observed from `await` plots, or because no such day exists in which faults are completely absent), the influence of faults is only to dampen alarms on the faulty components; non-faulty components remain unaffected. Thus, we recommend that training data should be sufficiently free from observable problems that an operator would feel comfortable operating the cluster indefinitely in its state at the time of training.

4.2.2 Algorithm Refinements

A peer-comparison algorithm requires the use of some measure that captures the similarity and the dissimilarity in the respective behaviors of peer components. A good measure, from a diagnosis viewpoint, is one that captures the differences between a faulty component and its non-faulty peer in a statistically significant way. In our explorations with Intrepid, we have sought to use robust similarity/dissimilarity measures that are improvements over the ones that we used in [17].

The first of these improvements is the method of histogram-bin selection. In [17] we used Sturges' rule [31] to base the number of histogram bins on $WinSize$. Under both faulty and fault-free scenarios (particularly where a LUN exhibits a small asymmetry), Sturges' rule creates histograms where all data is contained in the first and last bins. Thus, the amount of asymmetry of a specific LUN relative to the variance of all LUNs is lost and not represented in the histogram. In contrast, the Freedman–Diaconis rule selects bin size as a function of the interquartile range (IQR), a robust measure of variance uninfluenced by a small number of outliers. Thus, the number of bins in each histogram adapts to ensure an accurate histogram representation of asymmetries that exceeds natural variance.

One notable concern of the Freedman–Diaconis rule is the lack of a limit on the number of bins. Should a metric include outliers that are orders of magnitude larger than the IQR, then, the Freedman–Diaconis rule will generate infeasibly large histograms, which is problematic as the analysis time and memory requirements both scale linearly with the number of bins. While we found this to not typically be an issue with the `await` metric, `wr_sec` outliers would (attempt) to generate histograms with more than 18 million bins. For diagnosis on Intrepid's storage system, we use a bin limit of 1000, which is the 99th, 91st, and 87th percentiles for `await`, `rd_sec`, and `wr_sec` respectively, and results in a worst-case (all generated with 1000 bins) histogram-computation time that is only twice the average.

The second improvement of this algorithm is its use of CDF distances as a similarity/dissimilarity measure, instead of the Probability Density Functions (PDFs) distances as we used in [17]. Specifically, in [17], we used a symmetric version of Kullback-Leibler (KL) divergence [9] to compute distance using histogram approximations of metric PDFs. This comparison works well when two histograms overlap (i.e., many of their data points lie in overlapping bins). However, where two

```
Date.Hour:   Value PG:LUN-server
20110417.00:     7 2:18.61.102     6 2:15.65.74      5 2:11.55.48      5 2:12.48.49
20110417.01:    14 2:15.65.74      9 2:16.51.84      8 1:6.39.8        8 1:10.03.36
20110417.02:    19 2:15.65.74     17 2:16.51.84     15 2:10.50.35     15 2:21.56.121
20110417.03:    25 2:16.51.84     15 2:15.65.74     14 2:21.56.121    13 2:10.50.35
20110417.04:    33 2:16.51.84     20 2:15.65.74     15 2:21.56.121    13 2:10.50.35
20110417.05:    41 2:16.51.84     22 2:15.65.74     13 2:19.53.110    10 1:16.30.87
```

Figure 5: Example list of persistently anomalous LUNs. Each hour (row) specifies the most persistent anomalies (columns of accumulator value, peer-group, and LUN-server designation), ordered by decreasing accumulator value.

Feature	Test Bench	Intrepid	Rationale
Separating upgraded components	✗	✓	Tolerates weighted I/O on recently added storage capacity; addresses challenge #3.
Fundamental component for analysis	LUNs	LUN-server attachments	Provides views of LUN utilization across redundant components; improves problem localization; addresses challenges #1 and #5.
`cycle` daemon	✗	✓	Enables continuous instrumentation with `sadc`; addresses challenge #4.
Downsampling	✗	1 s → 15 s	Tolerates intermittent data, reduces resource requirements; addresses challenge #1.
Histogram bin selection	Sturges' rule	Freedman–Diaconis rule	Provides accurate representation of asymmetries; improves diagnostic accuracy.
Distance metric	KL Divergence (PDF)	Cumulative Distance (CDF)	Accurate distance for non-overlapping histograms; improves diagnostic accuracy.
Persistence Ordering	✗	✓	Highlight components with long-term problems; addresses challenge #6.

Table 2: Improvements to diagnosis approach as compared to previous work [17].

histograms are entirely non-overlapping (i.e., their data points lie entirely in non-overlapping bins in distinct regions of their PDFs), the KL divergence does not include a measure of the distance between non-zero PDF regions. In contrast, the distance between two metric CDFs *does* measure the distance between the non-zero PDF regions, which captures the degree of the LUN's asymmetry.

4.3 Persistence Ordering

While anomaly detection provides us with a reliable account of instantaneously anomalous LUNs, systems of comparable size to Intrepid with thousands of analyzed components, nearly always exhibit one or more anomalies for any given time window, even in the absence of an observable performance degradation.

Motivation. The fact that anomalies "always exist" is a key fact that requires us to alter our focus as we graduate from test-bench experiments to performing problem diagnosis on real systems. In our test-bench work, instantaneous anomalies were rare and either reflected the presence of our injected faults (which we aimed to observe), or the occurrence of false positive (which we aimed to avoid). However, in Intrepid, "spurious" anomalies (even with anomaly filtering) are common enough that we simply cannot raise alarms on each. It is also not possible to completely avoid the alarms through tweaking of analysis parameters (filter width, $WinSize$ and $WinShift$, etc.).

Investigating these spurious anomalies, we find that many are clear instances of transient asymmetries in our raw instrumentation data, due to occasional but regular events where behavior deviates across LUNs. Thus, for Intrepid, we focus our concern on locating system components that demonstrate long-term, or *persistent* anomalies, because they are suggestive of possible impending component failures or problems that might require manual intervention in order to resolve.

Algorithm. To locate persistent anomalies, it is necessary for us to order the list of anomalous LUNs by a measure of their impact on overall performance. To do so, we maintain a positive-value accumulator for every LUN in which we add one ($+1$) for each window where the LUN is anomalous, and subtract one (-1, and only if the accumulator is > 0) for each window where the LUN is not. We then present to the operator a list of *persistently* anomalous LUNs that are ordered by decreasing accumulator value, i.e., the top-most LUN in the list is that which has the most number of anomalous windows in its recent history. See Figure 5 for an example list.

4.4 Revisiting our Challenges

Table 2 provides a summary of the changes to our approach as we moved from our test-bench environment to performing problem diagnosis in a large-scale storage system. This combination of changes both adequately addresses the challenges of targeting Intrepid's storage system, and also improves the underlying algorithm.

Analysis Step	Runtime	Memory
Extract activity file contents (`dump`)	1.8 h	< 10 MB
Downsample and tabulate metrics (`table`)	7.1 h	1.1 GB
Anomaly Detection (`diagprep`)	49 m	6.1 GB
Persistence Ordering (`diagnose`)	1.6 s	36 MB
Total	9.7 h	6.1 GB

Table 3: Resources used in analysis of `await`, for the first four peer groups of the 2011-05-09 data set.

4.5 Analysis Resource Requirements

In this section, we discuss the resources (data volume, computation time, and memory) requirements for the analysis of Intrepid's storage system.

Data Volume. The activity files generated by `sadc` at a sampling interval of 1 s, when compressed with `xz` [8] at preset level `-1`, generate a data volume of approximately 10 MB per file server, per day. The median-size data set (for the day 2011-05-09) has a total (includes all file servers) compressed size of 1.3 GB. In total, we have collected 624 GB in data sets for 474 days.

Runtime. We perform our analysis offline, on a separate cluster consisting of 2.4 GHz dual-core AMD Opteron 1220 machines, each with 4 GB RAM. Table 3 lists our analysis runtime for the `await` metric, when utilizing a single 2.4 GHz Opteron core, for each step of our analysis for the first four peer groups of the median-size data set. Because each data set (consisting of 24 hours of instrumentation data) takes approximately 9.7 h to analyze, we are able to keep up with data input.

We note that the two steps of our analysis that dominate runtime—extracting activity file contents (which is performed on all metrics at once), and downsampling and tabulation of metrics (includes `await` only)—take long due to our sampling at a 1 s interval. We use the 1 s sample rate for archival purposes, as it is the highest sample rate `sadc` supports. However, we could sample at a 15 s rate directly and forgo the downsampling process, which reduces the extraction time in Table 3 by a factor of 15 and tabulation time to 31 m, yielding a total runtime of approximately 1.4h.

Algorithm Scalability. Our CDF distances are generated through the pairwise comparison of histograms is $O(n^2)$ where n is the number of LUNs in each peer group. Because our four peer groups consist of two sets of 768 and 384 LUNs, and our CDF distances are symmetric, we must perform a total of 736,128 histogram comparisons for each analysis window. In practice, we find that our CDF distances are generated quickly, as illustrated by our Anomaly Detection runtime of 49 m for 192 analysis windows (24 hours of data). Thus, we do not see our pairwise algorithm to be an immediate threat to scalability in terms of analysis runtime. We have also proposed [17] an alternative approach to enable $O(n)$ scalability, but found it unnecessary for use in Intrepid.

Memory Utilization. Table 3 also lists the maximum amount of memory used by each step of our analysis. We use the analysis process' Resident Set Size (RSS) plus any additional used swap memory to determine memory utilization. The most memory-intensive step of our analysis is Anomaly Detection. Our static memory costs come from the need to store the tabulated raw metrics, moving-average-filtered metrics, and a mapping of LUNs to CDF distances, each of which uses 101 MB of memory. Within each analysis window, we must generate histograms for each of the 2,304 LUNs in all four of our peer groups. With a maximum of 1000 bins, all of the histograms occupy at most 8.8 MB of memory. We also generate 736,128 CDF distances, which occupy 2.8 MB per window. However, we must maintain the CDF distances across all 192 analysis windows for a given 24-hour data set, comprising a total of 539 MB. Using R's [25] default garbage collection parameters, we find that the steady-state memory use while generating CDF distances to be 1.1 GB. The maximum use of 6.1 GB is transient, happening at the very end when our CDF distances are written out to file. With these memory requirements, we are able to analyze two metrics simultaneously on each of our dual-core machines with 4 GB RAM, using swap memory to back the additional 2–4 GB when writing CDF distances to file.

Diagnosis Latency. Our minimum diagnosis latency, that is, the time from the incident of an event to the time of its earliest report as an anomaly is 22.5 minutes. This figure is derived from our (i) performing analysis at a sampling interval of 15 s, (ii) analyzing in time windows shifted by 30 samples, and (iii) requiring that 3 out of the past 5 windows exhibits anomalous behavior before reporting the LUN itself as anomalous:

$$15\,{}^{\mathrm{s}}/_{\mathrm{samples}} \times 30\,{}^{\mathrm{samples}}/_{\mathrm{window}} \times 3\,\mathrm{windows} = 22.5\,\mathrm{m}$$

This latency is an acceptable figure for a few reasons:

- As a tool to diagnose component-level problems when a system is otherwise performing correctly (although, perhaps at suboptimal performance and reduced availability), the system continues to operate usefully during the diagnosis period. Reductions in performance are generally tolerable until a problem can be resolved.
- This latency improves upon current practice in Intrepid's storage system, e.g., four-hour automated checks of storage controller availability and daily manual checks of controller logs for misbehavior.
- Gabel et al. [13], which targets a similar problem of finding component-level issues before they grow into full-system failures, uses a diagnosis interval of 24 hours, and thus, considers this latency an acceptable figure.

In circumstances where our diagnosis latency would be unacceptably long, lowering the configurable parameters (sample interval, $WinShift$, and Anomaly Filtering's k value) will reduce latency with a potential for increased reports of spurious anomalies, which itself may be an acceptable consequence if there is external indication that a problem exists, for which we may assist in localization. In general, systems that are sensitive to diagnosis latency may benefit from combining our approach with problem-specific ones (e.g., heartbeats, SLAs, threshold limits, and component-specific monitoring) so as to complement each other in problem coverage.

5 Evaluation: Case Study

Having migrated our analysis approach to meeting the challenges of problem diagnosis on a large-scale system, we perform a case study of Intrepid over a 474-day period from April 13th, 2011 through July 31st, 2012. We use the second day (April 14th, 2011) as our only "training day" for threshold selection.

In this study we analyze both "old" and "new" LUN sets, for the first two LUN-server attachment priority groups. This enables us to observe "lost attachment" faults both with zero/missing data from the lost attachment with the primary file server (priority group 1), and with the new, non-peer workload on the attachment with the secondary file server (priority group 2). We note that, although we do not explicitly study priority groups 3–8, we have observed sufficient file server faults to require use of tertiary and subsequent file server attachments.

5.1 Method of Evaluation

After collecting instrumentation data from Intrepid's file servers, we perform the problem diagnosis algorithm described in § 4.2 on the `await` metric, generating a list of the top 100 persistently anomalous LUNs for each hour of the study.

In generating this list, we use a feedback mechanism that approximates the behavior of an operator using this system in real-time. For every period, we consider the top-most persistent anomaly, and if it has sufficient persistence (e.g., an accumulator value in excess of 100, which indicates that the anomaly has been present for at least half a day, but lower values may be accepted given other contextual factors such as multiple LUNs on the same controller exhibiting anomalies simultaneously), then, we investigate that LUN's instrumentation data and storage-controller logs to determine if there is an outstanding performance problem on the LUN, its storage controller, file-server attachments, or attached file servers.

At the time that a problem is remedied (which we determine through instrumentation data and logs, but would be recorded by an operator after performing the restorative operation), we zero the accumulator for the affected LUN to avoid masking subsequent problems during the anomaly's "wind-down" time (the time during which the algorithm would continually subtract one from the former anomaly's accumulated value until zero is reached). For anomalies that persist for more than a few days before being repaired, we regenerate the persistent-anomaly list with the affected LUNs removed from the list, and check for additional anomalies that indicate a second problem exists concurrently. If a second problem does exist, we repeat this process.

5.2 Observed Incidents

Using our diagnosis approach, we have uncovered a variety of issues that manifested on Intrepid's storage system performance metrics (and that, therefore, we suspect to be performance problems). Our uncovering of these issues was done through our independent analysis of the instrumentation data, with subsequent corroboration of the incident with system logs, operators, and manual inspection of raw metrics. We have grouped these incidents into three categories.

5.2.1 Lost Attachments

We use the *lost attachments* category to describe any problem whereby a file server no longer routes I/O for a particular LUN, i.e., the attachment between that LUN-server pair is "lost". Of particular concern are lost primary (or highest priority) attachments as it forces clients to reroute I/O through the secondary file server, which then sees a doubling of its workload. Lost attachments of other priorities may still be significant events, but they are not necessarily performance impacting as they are infrequently used for I/O. We observe four general problems that result in lost attachments: (i) failed (or simply unavailable) file servers, (ii) failed storage controllers, (iii) misconfigured components, and (iv) temporary "bad state" problems that usually resolve themselves on reboot.

Failed Events. Table 4 lists the observed down file-server and failed storage-controller events. The *incident time* is the time at which a problem is observed in instrumentation data or controller logs. *Diagnosis latency* is the elapsed time between *incident time* and when we identify the problem using our method of evaluation (see § 5.1). *Recovery latency* is the elapsed time between *incident time* and when our analysis observes the problems to be recovered by Intrepid's operators. *Device* is the component in the system that is physically closest to the origin of the problem, while the incident's observed manifestation is described in *description*. In particular, "missing data" refers to instrumentation data no longer being available for the specified LUN-server attachment due to the disappearance of the LUN's block device on

Incident Time	Diagnosis Latency	Recovery Latency	Device	Description
2011-07-14 00:00	1.0 h	25.8 d	ddn19a	Controller failed on reboot; missing data on 19.00.105.
2011-08-01 19:00	17.0 h	8.9 d	fs16	File server down; observed load on secondary (fs9).
2011-08-15 07:31	29 m	16.5 d	fs24	File server down; observed load on secondary (fs17).
2011-09-05 03:23	8.6 h	18.5 d	ddn20b	Controller manually failed; missing data on 20.00.117.
2011-09-11 03:22	11.6 h	35.6 h	fs25	File server down; observed load on secondary (fs26).
2011-10-03 03:09	12.9 h	42.5 d	ddn11a	Controller failed on reboot; observed load on secondary (fs41).
2011-10-17 16:57	22.1 h	28.0 d	ddn12a,20a,21a	Controllers manually failed; observed loads on secondaries (fs53,115,125).
2012-06-14 22:26	7.6 h	3.9 d	ddn8a	Controller manually failed; observed load on secondary (fs20).

Table 4: Storage controller failures and down file server lost attachment events.

Incident Time	Diagnosis Latency	Recovery Latency	Device	Description
2011-05-18 00:21	39 m	49.9 d	fs49,50,53,54	Extremely high await (up to 103 s) due to ddn12 resetting all LUNs, results in GPFS timeouts when accessing some LUNs, which remain unavailable until the affected file servers are rebooted; observed "0" await on 12.48.49.
2011-08-08 19:36	8.4 h	21.9 h	ddn19a	Servers unable to access some or all LUNs due to controller misconfiguration (disabled cache coherency); observed "0" await on 19.50.107.
2011-11-14 19:41	7.6 h	3.9 d	ddn14,18	Cache coherency fails to establish between coupled controllers after reboot, restricting LUN availability to servers; missing data on 14.41.67 and 18.37.103.
2012-03-05 17:50	3.2 h	4.2 d	fs56	GPFS service not available after file server reboot, unknown reason; observed loads on secondary (fs49).
2012-05-10 03:00	9.0 h	4.7 d	ddn16b	LUNs inaccessible from fs84,88, unknown reason; fixed on controller reboot; missing data on 16.59.84.
2012-06-13 03:00	3.0 h	8.7 d	ddn11a	LUNs inaccessible from fs42,45,46, unknown reason; missing data on 11.69.46.

Table 5: Misconfigured component and temporary "bad state" lost attachment events.

that file server, while a "0" value means the block device is still present, but not utilized for I/O.

The lengthy *recovery latency* for each of these *failed* events is due to the fact that all (except for fs25) required hardware replacements to be performed, usually during Intrepid's biweekly maintenance window, and perhaps even after consultation and troubleshooting of the component with its vendor. At present, Intrepid's operators discover these problems with syslog monitoring (for file servers) and by polling storage-controller status every four hours. Our *diagnosis latency* is high for file-server issues as we depend on the presence of a workload to diagnose traffic to the secondary attachment. Normally these issues would be observed sooner through missing values, except the instrumentation data itself comes from the down file server, and so, is missing in its entirety at the time of the problem (although the missing instrumentation data is a trivial sign that the file server is not in operation). In general, *failed* events, although they can be diagnosed independently, are important for analysis because they are among the longest-duration, numerous-LUN-impacting problems observed in the system.

Misconfiguration and Bad State Events. Table 5 lists the observed misconfiguration and temporary "bad state" events that result in lost attachments. We explain the two cache-coherency events as follows: Each storage array consists of two coupled storage-controllers, each attached to four different file servers, and both of which are able to provide access to attached disk arrays in the event of one controller's failure. However, when both controllers are in healthy operation, they may run in either cache-coherent or non-coherent modes. In cache-coherent mode, all LUNs may be accessed by both controllers (and thus, all eight file servers) simultaneously, as they are expected to by the GPFS-cluster configuration. However, should the controllers enter non-coherent mode (due to misconfiguration or a previous controller problem), then they can only access arrays "owned" by the respective controller, restricting four of the eight file servers from accessing some subset of the controllers' LUNs.

Cascaded Failure. The most interesting example in the "bad state" events is the GPFS timeouts of May 18th, 2011, a cascaded failure that went unnoticed by Intrepid operators for some time. Until the time of the incident, the ddn12 controllers were suffering from multiple, frequent disk issues (e.g., I/O timeouts) when the controller performed 71 "LUN resets". At this time, the controller delayed responses to incoming I/O requests for up to

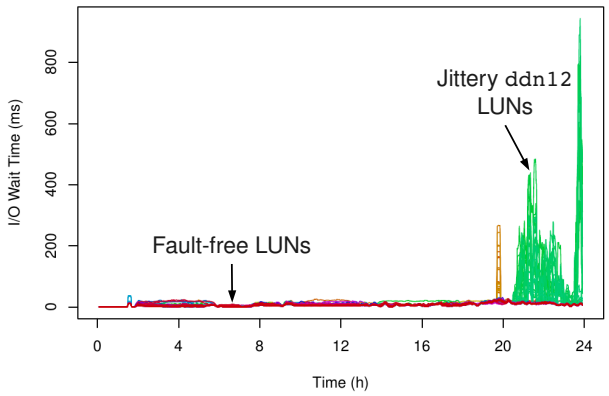

Figure 6: I/O wait time jitter experienced by `ddn12` LUNs during the May 17th, 2011 drawer error event.

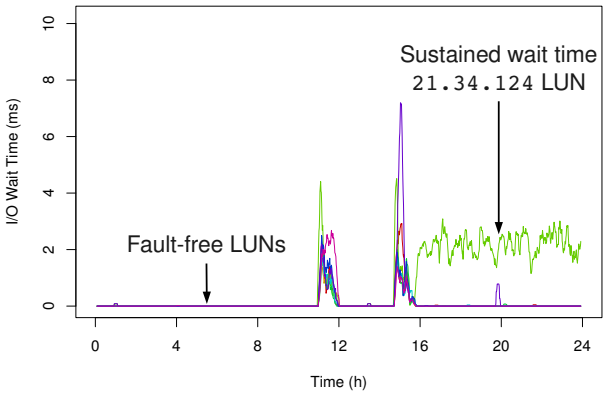

Figure 7: Sustained I/O wait time experienced by `21.34.124` during the June 25th single LUN event.

103 s, causing three of the file servers to timeout their outstanding I/Os and refuse further access to the affected LUNs. Interestingly, while the controller and LUNs remain in operation, the affected file servers continue to abandon access for the duration of 50 days until they are rebooted, at which point the problem is resolved. This particular issue highlights the main benefit of our holistic peer-comparison approach. By having a complete view of the storage system, our diagnosis algorithm is able to locate problems that otherwise escape manual debugging and purpose-specific automated troubleshooting (i.e., scripts written to detect specific problems).

5.2.2 Drawer Errors

A drawer error is an event where a storage controller finds errors, usually I/O and "stuck link" errors on many disks within a single disk drawer. These errors can become very frequent, occurring every few seconds, adding considerable jitter to I/O operations (see Figure 6). Table 6 lists four observed instance of drawer errors, which are fairly similar in their diagnosis characteristics. Drawer errors are visible to operators as a series of many verbose log messages. Operators resolve these errors by forcibly failing every disk in the drawer, rebooting the drawer, then reinserting all the disks into their respective arrays, which are recovered quickly via journal recovery.

5.2.3 Single LUN Events

Single LUN events are instances where a single LUN exhibits considerable I/O wait time (`await`) for as little as a few hours, or as long as many days. Table 7 lists five such events although as many as 40 have been observed to varying extents during our analysis.

These events can vary considerably in their behavior, and Table 7 provides a representative sample. Occasionally, the event will be accompanied by one or more controller-log messages that suggests that one or more spindles in the LUN's disk array is failing, e.g., the June 18th event is accompanied with a message stating that the controller recovered an "$8+2$" parity error. Single LUN events may correspond to single-LUN workloads, and thus, would manifest in one of the throughput metrics (`rd_sec` or `wr_sec`) in addition to `await`. Conversely, the June 25th event in Table 7 manifests in `await` in the *absence* of an observable workload (see Figure 7), perhaps suggesting that there is a load internal to the storage controller or array that causes externally-visible delay. Unfortunately, since storage-controller logs report little on most of our single LUN events, it is difficult to obtain a better understanding of specific causes of these events.

5.3 Alternative Distance Measures

Our use of CDF distances as the distance measure for our peer-comparison algorithm is motivated by its ability to capture the differences in performance metrics between a faulty component and its non-faulty peer. Specifically, CDF distances capture asymmetries in a metric's value (relative to the metric's variance across all LUNs), as well as asymmetries in a metric's shape (i.e., a periodic or increasing/decreasing metric vs. a flat or unchanging metric). The use of CDF distances does require pairwise comparison of histograms, and thus, is $O(n^2)$ where n is the number of LUNs in each peer group. While we have demonstrated that the use of pairwise comparisons is not an immediate threat to scalability (see § 4.5), it is illustrative to compare CDF distances to alternative, computationally-simpler, $O(n)$ distance measures.

The two alternative distance measures we investigate are *median* and *thresh*. For both measures, we use the same Anomaly Detection and Persistence Ordering algorithms as described in § 4.2 and § 4.3, including all windowing, filtering, and their associated parameters. For each time window, instead of generating histograms we use one of our alternative measures to gen-

Incident Time	Diagnosis Latency	Duration	Device	Description
2011-05-17 20:30	3.5 h	3.9 h	ddn12	Jittery await for 60 LUNs due to frequent "G" drawer errors.
2011-06-20 18:30	1.5 h	48.8 h	ddn12	Jittery await for 37 LUNs due to frequent "G" drawer errors.
2011-09-08 02:00	12.0 h	27.0 h	ddn19	Jittery await for 54 LUNs due to frequent "A" drawer errors.
2012-01-16 19:30	3.5 h	24.0 h	ddn16	Jittery await for 11 LUNs due to frequent "D" drawer errors.

Table 6: Drawer error events.

Incident Time	Diagnosis Latency	Duration	Device	Description
2011-06-18 08:00	18.0 h	10.5 d	15.37.78	Sustained above average await; recovered parity errors.
2011-08-18 20:00	26.0 h	79.0 h	19.12.109	Sustained above average await; until workload completes.
2011-09-25 04:00	20.0 h	4.3 d	11.12.45	Sustained above average await; unknown reason.
2012-04-19 12:00	38.0 h	7.2 d	9.04.29	Sustained above average await; unknown reason.
2012-06-25 16:00	8.0 h	6.6 d	21.34.124	Sustained await in absence of workload; unknown reason.

Table 7: Single LUN events.

erate, for each LUN, a scalar distance value from the set of *WinSize* samples. For *median*, we generate a median time-series, m by computing for each sample, the median value across all LUNs within a peer group. We then compute each LUN's scalar distance as the sum of the distances between that LUN's metric value x and the median value for each of the *WinSize* samples: $d(x,m) = \sum_{i=0}^{WinSize} |x(i) - m(i)|$. We then flag a LUN as anomalous over a window if its scalar distance exceeds a predefined threshold, which is selected using the approach described in § 4.2.1.

We follow the same procedure for *thresh*, except that each LUN's scalar "distance" value is calculated simply as the maximum metric value x among the *WinSize* samples: $d(x) = \max x(i) \,|_{i=0}^{WinSize}$. Here, *thresh* is neither truly a measure of *distance*, nor is it being used to perform peer-comparison. Instead we use the *thresh* measure to implement the traditional "metric exceeds alarm-threshold value" within our anomaly detection framework, i.e., an anomalous window using *thresh* indicates that the metric exceeded twice the highest-observed value during the training period for at least one sample.

Performing a meaningful comparison of the *median* and *thresh* measures against CDF distances is challenging with production systems like Intrepid, where our evaluation involves some expert decision making and where we lack ground-truth data. For example, while the events enumerated in Tables 4–7 represent the most significant issues observed in our case study, we know there exists many issues of lesser significance (especially single LUN events) that we have not enumerated. Thus it is not feasible to provide traditional accuracy (true- and false-positive) rates as we have in our test-bench experiments. Instead, we compare the ability of the *median* and *thresh* measures to observe the set of events discovered using CDF distances (listed in Tables 4–7), by following the evaluation procedure described in § 5.1 for the days during which these events occur.

Event Type	CDF	Median	Thresh
Controller failure	5	5	5
File server down	3	2	3
Misconfiguration / bad state	6	6	5
Drawer error	4	3	4
Single LUN	5+	2	1

Table 8: Number of events observed with each distance measure (CDF distances, *median*, and *thresh*).

5.3.1 Comparison of Observations

Table 8 lists the number of we events observe with the alternative distance measures, *median* and *thresh*, as compared to the total events observed with CDF distances. Both *median* and *thresh* measures are able to observe all five failed storage-controller events, as well as most down file-server and misconfiguration/"bad state" events. Each of these events are characterized by missing data on the LUN's primary attachment, and the appearance of a load on the LUN's normally-unused secondary attachment. Unlike CDF distances, neither *median* nor *thresh* measures directly account for missing data, however these events are still observed through the presence of secondary-attachment loads. As the non-primary attachments of LUNs are rarely used, these secondary-attachment loads are significant enough to contribute to considerable distance from the (near zero) median and to exceed any value observed during fault-free training. Both measures are also able to observe most drawer errors as these events exhibit considerable peak await that exceed both the median value and the maximum-observed value during training.

Controller Misconfiguration. For the August 8th, 2011 controller misconfiguration event, a zero await value is observed on the affected LUNs' primary attachments for the duration of the event, which is observed by the *median* measure. However, this particular event also results in zero await on the LUNs' secondary attachments, which are also affected, pushing the load onto the

LUNs' tertiary attachments. As we only analyze each LUN's first two priority groups, the load on the tertiary attachment (and the event itself) goes unobserved. Thus, the *thresh* measure requires analysis of all priority groups to locate "missing" loads that are otherwise directly observed with peer-comparison-based measures.

Single LUN Events. Two single LUN events go unobserved by *median* as their manifestations in increased `await` are not sufficiently higher than their medians to result in persistent anomalies. Four of the events go unobserved by *thresh* as `await` never exceeds its maximum value observed during training, except during the June 25th, 2012 incident on a (normally-unused) secondary attachment where sustained `await` is observed in absence of a workload.

5.3.2 Server Workloads.

The remaining three events that escape observation by *median* (a down file-server, drawer error, and single LUN events) are each due to the same confounding issue. As described in § 3.3.2, shared storage is normally prioritized such that GPFS clients only use the highest-priority available attachment. However, workloads issued by GPFS file servers themselves preferentially make use of their own LUN attachments, regardless of priority, to avoid creating additional LAN traffic. Thus, for server-issued workloads, we observe loads on each (e.g., 48) of the server's attachments, which span all priority groups, as well as loads on each (e.g., 720) of the primary attachments for LUNs that are not directly attached to those servers. Such workloads, if significant enough, would result in anomalies on each (e.g., 42) of the non-primary attachments.

In practice, Intrepid's storage system does not run significant, long-running workloads on the file servers, so this complication is usually avoided. The exception is that GPFS itself occasionally issues very low-intensity, long-running (multiple day) workloads from an apparently-random file server. These workloads are of such low intensity (throughput < 10 kB/s, `await` < 1.0 ms, both per LUN) that their `await` values rarely exceed our CDF distance algorithm's histogram *BinSize*s, and thus, are regarded as noise. However, server-workload `await` values on non-primary attachments do exceed the (zero) median value, and thus, do contribute to *median* anomalies. The result is that the presence of a server workload during an analysis window often exhibits a greater persistence value than actual problems, which confounds our analysis with the *median* measure. Thus, reliable use of the *median* measure requires an additional analysis step to ignore anomalies that appear across all attachments for a particular file server.

Event Type	Median (h)	Thresh (h)
Controller failure	−7, 0, 5, 9, 12	−10, 0, 4, 5, 9
File server down	0, 1	−8, 0, 6
Misconfiguration / bad state	−8, −3, 0, 4, 6, 7	−3, −1, 0, 3, 7
Drawer error	−2, −1, 5	−2, −2, −1, 0
Single LUN	−5, 6	−4

Table 9: Differences in diagnosis latencies for events observed with alternative measures, as compared to CDF.

5.3.3 Comparison of Latencies

Table 9 lists the differences in diagnosis latencies for events observed with the alternative distance measures, *median* and *thresh*, as compared to the diagnosis latencies observed with CDF distances. Negative values indicate that the alternative measure (*median* or *thresh*) observed the event before CDF distances, while positive values indicate that the alternative measure observed the event after. Differences are indicated in integer units as our reporting for the case study is hourly (see Figure 5).

With a mean 1.6 h and median 0.5 h increased latency for *median*, and a mean 0.2 h and median 0 h increased latency for *thresh*, diagnosis latency among all three distance measures are comparable. However, for specific events, latencies can vary as much as twelve hours between measures, suggesting that simultaneous use of multiple measures may be helpful to reduce overall diagnosis latency.

6 Experiences and Insights

In preparing for our case study of Intrepid's storage system, we made improvements to our diagnosis approach to address the challenges outlined in § 3. However, in the course of our instrumentation and case study, we encountered a variety of pragmatic issues, and we share our experiences and insights with them here.

Clock synchronization. Our diagnosis algorithm requires clocks to be reasonably synchronized across file servers so that we may peer-compare data from the same time intervals. In our test-bench experiments [17], we used NTP to synchronize clocks at the start of our experiments, but disabled the NTP daemon so as to avoid clock adjustments during the experiments themselves. Intrepid's file servers also run NTP daemons; however, clock adjustments can and do happen during our `sadc` instrumentation. This results in occasional "missing" data samples where the clock adjusts forward, or the occasional "repeat" sample where the clock adjusts backwards. When tabulating data for analysis, we represent missing samples with R's `NA` (missing) value, and repeated samples are overwritten with the latest recorded in the activity file. In general, our diagnosis is insensitive to minor clock adjustments and other delays that may result in missing samples, but it is a situation we initially encountered in our `table` script.

Discussion on timestamps. Our activity files are recorded with filenames containing a modified[2] ISO 8601-formatted [19][3] UTC timestamp that corresponds to the time of the first recorded sample in the file. For example, `fs1-20110509T000005Z.sa.xz` is the activity file collected from file server `fs1`, with the first sample recorded at 00:00:05 UTC on 2011-05-09. In general, we recommend the use of ISO 8601-formatted UTC timestamps for filenames and logging where possible, as they provide the following benefits:

- Human readable (as opposed to Unix time).
- Ensures lexicographical sorting (e.g., of activity files) preserves the chronological order (of records).
- Contains no whitespace, so is easily read as a field by awk, R's `read.table`, etc.
- Encodes time zone as a numeric offset; "`Z`" for UTC.

With regard to time zones, ISO 8601's explicit encoding of them is particularly helpful in avoiding surprises when interpreting timestamps. It is an obvious problem if some components of a system report different time zones than others without expressing their respective zones in timestamps. However, even when all components use the same time zone (as Intrepid uses UTC), offline analysis may use timestamp parsing routines that interpret timestamps without an explicit time-zone designation in the local time zone of the analysis machine (which, in our case, is US Eastern).

A more troubling problem with implicit time zones is that any timestamp recorded during the "repeating" hour of transition from daylight savings time to standard time (e.g., 1 am CDT to 1 am CST) are ambiguous. Although this problem happens only once a year, it causes difficulty in correlating anomalies observed during this hour with event logs from system components that lack time zone designations. Alternatively, when when components do encode time zones in timestamps, ISO 8601's use of numeric offsets makes it easy to convert between time zones without needing to consult a time zone database to locate the policies (e.g, daylight savings transition dates) behind time-zone abbreviations.

In summary, ISO 8601 enables easy handling of human readable timestamps without having to work-around edge cases inevitable when performing continuous instrumentation and monitoring of system activity. "Seconds elapsed since epoch" time (e.g., Unix time) works well as a non-human readable alternative as long as the epoch is unambiguous. `sadc` records timestamps in Unix time, and we have had no trouble with them.

[2] We remove colons to ensure compatibility with file systems that use colons as path separators.

[3] RFC 3339 is an "Internet profile of the ISO 8601 standard," that we cite due to its free availability and applicability to computer systems.

Absence of data. One of the surprising outcomes of our case study is that the *absence of*, or "missing data" where it is otherwise expected among its peers, is the primary indication of problem in five (seven if also including "0" data) of the studied events. This result reflects on the effectiveness of peer-comparison approaches for problem diagnosis as they highlight differences in behavior across components. In contrast, approaches that rely on thresholding of raw metric values may not indicate that problems were present in these scenarios.

Separation of instrumentation from analysis. Our diagnosis for Intrepid's storage system consists of simple, online instrumentation, in conjunction with more complex, offline analysis. We have found this separation of online instrumentation and offline analysis to be beneficial in our transition to Intrepid. Our instrumentation, consisting of a well-known daemon (`sadc`), and a small, C-language auditable tool (`cycle`), have few external dependencies and negligible overhead, both of which are important properties to operators considering deployment on a production system. In contrast, our analysis has significant resource requirements and external dependencies (e.g., the R language runtime and associated libraries), and so is better suited to run on a dedicated machine isolated from the rest of the system. We find that this separation provides an appropriate balance in stability of instrumentation and flexibility in analysis, such that, as we consider "near real-time" diagnosis on Intrepid's storage system, we prefer to maintain the existing design instead of moving to a full-online approach.

7 Future Work

While our persistence-ordering approach works well to identify longer-term problems in Intrepid, there is a class of problems that escapes our current approach. Occasionally, storage controllers will greatly delay I/O processing in response to an internal problem, such as the "LUN resets" observed on `ddn12` in the May 18th, 2011 cascaded failure event. Although we observed this particular incident, in general, order-of-magnitude increases in I/O response times are not highlighted as we ignore the severity of an instantaneous anomaly. Thus, the development of an ordering method that factors in both the severity of instantaneous anomaly, as well as persistence, would be ideal in highlighting both classes of problems.

We also believe we could improve our current problem-diagnosis implementation (see § 5.1) to further increase its utility for systems as large as Intrepid. For instance, problems in storage controllers tend to manifest in a majority of their exported LUNs, and thus, a single problem can be responsible for as many as 50 of the most persistent anomalies. Extending our approach to recognize that these anomalous LUNs are a part of

the same storage controller, and thus, manifest the same problem, would allow us to collapse them into a single anomaly report. This in turn, would make it considerably easier to discover multiple problems that manifest in the same time period. Combining both of these improvements would certainly help us to expand our problem diagnosis and fault coverage.

8 Related Work

Problem Diagnosis in Production Systems. Gabel et al. [13] applies statistical latent fault detection using machine (e.g., performance) counters to a commercial search engine's indexing and service clusters, and finds that 20% of machine failures are preceded by an incubation period during which the machine deviates in behavior (analogous to our component-level problems) prior to system failure. Draco [18] diagnoses chronics in VoIP operations of a major ISP by, first, heuristically identifying user interactions likely to have failed, and second, identifying groups of properties that best explain the difference between failed and successful interactions. This approach is conceptually similar to ours in using a two-stage process to identify that (i) problems exists, and (ii) localizing them to the most problematic components. Theia [14] is a visualization tool that analyzes application-level logs and generates visual signatures of job performance, and is intended for use by users to locate and problems they experience in a production Hadoop cluster. Theia shares our philosophy of providing a tool to enable users (who act in a similar capacity to our operators) to quickly discover and locate component-level problems within these systems.

HPC Storage-System Characterization. Darshan [6] is a tool for low-overhead, scalable parallel I/O characterization of HPC workloads. Darshan shares our goal of minimal-overhead instrumentation by collecting aggregate statistical and timing information instead of traces in order to minimize runtime delay and data volume, which enables it to scale to leadership-class systems and be used in a "24/7", always-on manner. Carns et al. [5] combine multiple sources of instrumentation including OS-level storage device metrics, snapshots of file system contents characteristics (file sizes, ages, capacity, etc.), Darshan's application-level I/O behavior, and aggregate (system-wide) I/O bandwidth to characterize HPC storage system use and behavior. These characterization tools enable a better understanding of HPC application I/O and storage-system utilization, so that both may be optimized to maximize I/O efficiency. Our diagnosis approach is complementary to these efforts, in that it locates sources of acute performance imbalances and problems within the storage system, but assumes that applications are well-behaved and that the normal, balanced operation is optimal.

Trace-Based Problem Diagnosis. Many previous efforts have focused on path-based [1, 26, 3] and component-based [7, 20] approaches to problem diagnosis in Internet Services. Aguilera et al. [1] treats components in a distributed system as black-boxes, inferring paths by tracing RPC messages and detecting faults by identifying request-flow paths with abnormally long latencies. Pip [26] traces causal request-flows with tagged messages that are checked against programmer-specified expectations. Pip identifies requests and specific lines of code as faulty when they violate these expectations. Magpie [3] uses expert knowledge of event orderings to trace causal request-flows in a distributed system. Magpie then attributes system-resource utilizations (e.g. memory, CPU) to individual requests and clusters them by their resource-usage profiles to detect faulty requests. Pinpoint [7, 20] tags request flows through J2EE web-service systems, and, once a request is known to have failed, identifies the responsible request-processing components.

In HPC environments, Paradyn [22] and TAU [30] are profiling and tracing frameworks used in debugging parallel applications, and IOVIS [23] and Dinh [10] are retrofitted implementations of request-level tracing in PVFS. However, at present, there is limited request-level tracing available in production HPC storage deployments, and thus, we concentrate on a diagnosis approach that utilizes aggregate performance metrics as a readily-available, low-overhead instrumentation source.

Peer-comparison Based Approaches. Ganesha [24] seeks to diagnose performance-related problems in Hadoop by classifying slave nodes, via clustering of performance metrics, into behavioral profiles which are then peer-compared to indict nodes behaving anomalously. While the node-indictment methods are similar, our work peer-compares a limited set of performance metrics directly (without clustering). Bodik et at. [4] use fingerprints as a representation of state to generally diagnose previously-seen datacenter performance crises from SLA violations. Our work avoids using previously-observed faults, and instead relies on fault-free training data to capture expected performance deviations and peer-comparison to determine the presence, specifically, of storage performance problems. Wang et al. [32] analyzes metric distributions to identify RUBiS and Hadoop anomalies in entropy time-series. Our work also avoids the use of raw-metric thresholds by using peer-comparison to determine the degree of asymmetry between storage components, although we do threshold our distance measure to determine the existence of a fault. PeerWatch [16] peer-compares multiple instances of an application running across different virtual ma-

chines, and uses canonical correlation analysis to filter out workload changes and uncorrelated variables to find faults. We also use peer-comparison and bypass workload changes by looking for performance asymmetries, as opposed to analyzing raw metrics, across file servers.

Failures in HPC and Storage Systems. Studies of HPC and storage-system failures motivate our focus on diagnosing problems in storage-system hardware components. A study of failure data collected over nine-years from 22 HPC systems at Los Alamos National Laboratory (LANL) [28] finds that hardware is the largest root cause of failures at 30–60% across the different systems, with software the second-largest contributor at 5–24%, and 20–30% of failures having unknown cause. The large proportion of hardware failures motivates our concentration on hardware-related failures and performance problems. A field-based study of disk-replacement data covering 100,000 disks deployed in HPC and commercial storage systems [29] finds an annual disk replacement rate of 1–4% across the HPC storage systems, and also finds that hard disks are the most commonly replaced components (at 18–49% of the ten most frequently replaced components) in two of three studied storage systems. Given that disks dominate the number of distinct components in the Intrepid storage system, we expect that disk failures and (intermittent) disk performance problems comprise a significant proportion of hardware-related performance problems, and thus, are worthy of specific attention.

9 Conclusion

We presented our experiences of taking our problem diagnosis approach from proof-of-concept on a 12-server test-bench cluster, and making it work on Intrepid's production GPFS storage system. In doing so, we analyzed 2304 different component metrics across 474 days, and presented a 15-month case study of problems observed in Intrepid's storage system. We also shared our challenges, solutions, experiences, and insights towards performing continuous instrumentation and analysis. By diagnosing a variety of performance-related storage-system problems, we have shown the value of our approach for diagnosing problems in large-scale storage systems.

Acknowledgements

We thank our shepherd, Adam Oliner, for his comments that helped us to improve this paper. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE–AC02–06CH11357.

References

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, NY, Oct. 2003.

[2] Argonne National Laboratory. Blue Gene / P, Dec. 2007. `https://secure.flickr.com/photos/argonne/3323018571/`.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, San Francisco, CA, Dec. 2004.

[4] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen. Fingerprinting the datacenter: Automated classification of performance crises. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys '10)*, Paris, France, Apr. 2010.

[5] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage*, 7(3):8:1–8:26, Oct. 2011.

[6] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 characterization of petascale I/O workloads. In *Proceedings of the 1st Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS '09)*, New Orleans, LA, Sept. 2009.

[7] M. Y. Chen, E. Kıcıman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN '02)*, Bethesda, MD, June 2002.

[8] L. Collin. XZ utils, Apr. 2013. `http://tukaani.org/xz/`.

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

[10] T. D. Dinh. Tracing internal behavior in PVFS. Bachelorarbeit, Ruprecht-Karls-Universität Heidelberg, Heidelberg, Germany, Oct. 2009.

[11] M. Evans. Self-monitoring, analysis and reporting technology (S.M.A.R.T.). SFF Committee Specification SFF-8035i, Apr. 1996.

[12] D. Freedman and P. Diaconis. On the histogram as a density estimator: L2 theory. *Probability Theory and Related Fields*, 57(4):453–476, Dec. 1981.

[13] M. Gabel, A. Schuster, R.-G. Bachrach, and N. Bjørner. Latent fault detection in large scale services. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '12)*, Boston, MA, June 2012.

[14] E. Garduno, S. P. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. Theia: Visual signatures for problem diagnosis in large hadoop clusters. In *Proceedings of the 26th Large Installation System Administration Conference (LISA '12)*, Washington, DC, Nov. 2012.

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

[16] H. Kang, H. Chen, and G. Jiang. Peerwatch: a fault detection and diagnosis tool for virtualized consolidation systems. In *Proceedings of the 7th International Conference on Autonomic Computing (ICAC '10)*, Washington, DC, June 2010.

[17] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan. Black-box problem diagnosis in parallel file systems. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST '10)*, San Jose, CA, Feb. 2010.

[18] S. P. Kavulya, S. Daniels, K. Joshi, M. Hiltunen, R. Gandhi, and P. Narasimhan. Draco: Statistical diagnosis of chronic problems in large distributed systems. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '12)*, Boston, MA, June 2012.

[19] G. Klyne and C. Newman. Date and Time on the Internet: Timestamps. RFC 3339 (Proposed Standard), July 2002.

[20] 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.

[21] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock. I/O performance challenges at leadership scale. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*, Portland, OR, Nov. 2009.

[22] 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.

[23] C. Muelder, C. Sigovan, K.-L. Ma, J. Cope, S. Lang, K. Iskra, P. Beckman, and R. Ross. Visual analysis of I/O system behavior for high–end computing. In *Proceedings of the 3rd Workshop on Large-scale System and Application Performance (LSAP '11)*, San Jose, CA, June 2011.

[24] X. Pan, J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan. Ganesha: Black-box diagnosis of mapreduce systems. In *Proceedings of the 2nd Workshop on Hot Topics in Measurement & Modeling of Computer Systems (HotMetrics '09)*, Seattle, WA, June 2009.

[25] R Development Core Team. The R project for statistical computing, Apr. 2013. `http://www.r-project.org/`.

[26] 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 Symposium on Networked Systems Design and Implementation (NSDI '06)*, San Jose, CA, May 2006.

[27] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST '02)*, Montery, CA, Jan. 2002.

[28] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance-computing systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN '06)*, Philadelphia, PA, June 2006.

[29] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07)*, San Jose, CA, Feb. 2007.

[30] 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.

[31] H. A. Sturges. The choice of a class interval. *Journal of the American Statistical Association*, 21(153):65–66, Mar. 1926.

[32] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan. Online detection of utility cloud anomalies using metric distributions. In *Proceedings of 12th IEEE/IFIP Network Operations and Management Symposium (NOMS '10)*, Osaka, Japan, Apr. 2010.