

Gumshoe: Diagnosing Performance Problems in Replicated File-Systems

Soila Kavulya, Rajeev Gandhi and Priya Narasimhan
Electrical & Computer Engineering Department
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3891
spertet@ece.cmu.edu, rgandhi@ece.cmu.edu, priya@cs.cmu.edu

Abstract

Replicated file-systems can experience degraded performance that might not be adequately handled by the underlying fault-tolerant protocols. We describe the design and implementation of Gumshoe, a system that aims to diagnose performance problems in replicated file-systems. Gumshoe periodically gathers OS and protocol metrics and then analyzes these metrics to automatically localize the performance problem to the culprit node(s). We describe our results and experiences with problem diagnosis in two replicated file-systems (replicated-CoreFS and BFS) using two file-system benchmarks (Postmark and IOzone).

1 Introduction

File-systems can experience performance problems that can be hard to diagnose and isolate. Performance problems can arise from different system layers, e.g., resource exhaustion [13], misconfigurations of protocols [6]. Replicated file-systems exploit replication of the file-servers in an attempt to sustain file-server operation despite failures, and often exploit a fault-tolerant protocol [11] as a key underlying building-block. The protocol abstracts the complex details of fault-detection and provides the reliable, ordered message delivery required to maintain consistent state across the file-server replicas. These protocols perform reactive recovery, which means that they first wait to detect a failure (usually through timeouts), and then react to recover from the failure (typically by evicting the culprit node from the system). However, some kinds of problems (e.g., slow memory exhaustion) can cause the file-system’s performance to suffer, without immediately resulting in a crash fault that would be detectable via the protocol’s timeouts. Thus, these performance problems can either lie outside the fault-handling capabilities of these protocols, or can cause degraded performance for a significant period of time before the protocols’ timeouts are finally activated.

In this paper, we address the problem of automatically localizing the problem to the culprit node(s) and mitigating the impact of the problem on performance in replicated file-systems by isolating the culprit node(s) from the system. We describe the design, implementation and evaluation of Gumshoe, a system that aims to diagnose performance problems in replicated file-systems. As shown in Figure 1, Gumshoe’s OS- and protocol-level instrumentation collects traces of key performance metrics on every node in the system. Gumshoe’s algorithms analyze trends in the gathered traces to spot incipient anomalous behavior before a perceivable failure (replica crash) occurs.

The presence of file-server replicas in the system allows us perform diagnosis in a unique way—Gumshoe can execute a peer-comparison type of algorithm to correlate the traces gathered from replica nodes, effectively reducing false-positives due to application-level workload/mode changes (since all of the replicas can be expected to undergo these changes in a similar manner). Gumshoe can perform its diagnosis either online (i.e., as the file-system is running) or offline (i.e., post-processing on previously gathered traces). Gumshoe supports both a *black-box* approach of analyzing only OS-level metrics, as well as a *gray-box* approach of additionally analyzing protocol-level metrics.

We conducted our experiments with two replicated file-systems (replicated-CoreFS and BFS), each using a different underlying fault-tolerant protocol (Spread and Castro-Liskov BFT, respectively) and two file-system benchmarks (Postmark and IOzone). We inject real performance problems that are documented in the bug databases of publicly available file-systems. By targeting two replicated file-systems, each using a fault-tolerant protocol with a different architecture, we can assess whether the protocol’s architecture influences diagnosis. We evaluate Gumshoe’s black-box and gray-box diagnosis techniques, and compare Gumshoe’s capabilities with those of the underlying protocol’s failure-detection mechanisms. Under performance problems, we show that enabling Gumshoe increased the throughput for the Postmark workload by 4% to 12%.

2 Background

Replicated file-systems provide high availability by leveraging a fault-tolerant protocol [11] to provide guarantees of reliable, totally ordered message delivery between file-server replicas, even in asynchronous environments where message delays may be unbounded. The fault-tolerant protocol relies on timeouts to detect failures, and attempts to reduce all detected problems to node-group reconfigurations, e.g., a slow node, a lossy network all ultimately provoke a node-group membership change that evicts the suspected node from the node-group. We briefly describe the two replicated file-systems (replicated-CoreFS and BFS) that we target.

Although BFS is Byzantine-fault-tolerant and replicated-CoreFS is crash-fault-tolerant, our focus is on diagnosing performance problems whose impact can be compared across the two replicated file-systems. Malicious faults are currently outside the scope of this work.

2.1 Replicated CoreFS

CoreFS [19] is an open-source network file-system built on top of FUSE (Filesystem in User Space), an open-source kernel module that allows non-privileged users to create their own file systems without needing to write kernel code. We implement the state-machine replication [26] of the CoreFS file-server by exploiting the open-source, Spread token-ring protocol [4] which tolerates crash faults, communication faults and network partitions.

2.2 Byzantine-fault-tolerant NFS (BFS)

The Byzantine-fault-tolerant NFS service (BFS) is an open-source, replicated file-system built on top of the open-source Castro-Liskov BFT protocol [9]. BFS aims for highly available operation that tolerates Byzantine or arbitrary faults in the file-server replicas, and requires a greater degree of replication, i.e., $3f+1$ replicas to tolerate f faults. Unlike CoreFS (that we needed to replicate ourselves), BFS already supports replication. The underlying BFT protocol has a quorum-based architecture, which designates a primary file-server replica, along with backup replicas.

3 Problem Statement

Our research question was: *can we diagnose the culprit node in the face of a performance problem in a replicated file-system?* We describe our goals, non-goals and hypotheses in addressing this research question.

Goals. We required Gumshoe to satisfy the following specific goals.

- *Application-transparency.* Gumshoe should not require any modifications to the applications using the file-systems. Gumshoe’s operation should be independent of the file-system being targeted.
- *Minimize false-positive rate.* Gumshoe should be able to differentiate between anomalous behavior and legitimate workload changes (e.g., additional clients, increased request rate) or mode changes (e.g., peak load, system backup) in the system.
- *Minimize instrumentation overhead.* Gumshoe’s overheads for instrumentation and analysis should not adversely impact the file-system’s operation.

Non-Goals. Gumshoe aims for coarse-grained problem localization by identifying the culprit node(s). Fine-grained root-cause analysis, which would aim to identify the underlying fault or bug (possibly even down to the offending line of code), is outside Gumshoe’s current scope.

Hypotheses. We hypothesize that, under a performance problem in a file-system environment, OS-level (and, protocol-level) metrics will exhibit escalating anomalous behavior on the culprit node(s). We hypothesize that this escalation is observable for a period of time before the ultimate crash of the culprit node. Additionally, in replicated file-systems, we hypothesize that the statistical trends of these metrics: (i) will be similar (albeit with a possible lag) across problem-free nodes, and (ii) will differ on the culprit node, as compared to the problem-free nodes. The rest of the paper describes how we seek to validate these hypotheses through Gumshoe’s design, implementation and evaluation.

Assumptions. We assume that a majority of the file-server replica nodes exhibit problem-free behavior. We assume that all nodes are identical, dedicated machines, and that the physical clocks on the various nodes are synchronized (e.g., via NTP) so that we can extract time-stamped traces that can be temporally correlated across these nodes.

4 Design & Implementation

Our system consists of an instrumentation framework, an optional node-level anomaly-detector, and a dedicated diagnosis server. Each of these components is described in detail later, but we provide a synopsis of Gumshoe’s approach here. On each node that hosts a file-server replica, our instrumentation periodically collects time-series data of the OS- and protocol-level metrics (listed in Table 1) at runtime. These metrics are then fed, either processed or unprocessed, to our diagnosis server that compares the metric traces across nodes, effectively using an “odd-man-out” strategy to identify the culprit node as the one that deviates from the others in its performance metrics.

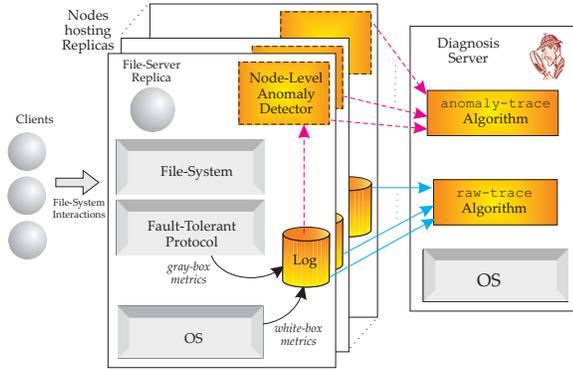


Figure 1. Gumshoe architecture.

Black-box metrics (from the OS)	CPU utilization CPU run-queue size pages in/out used, and free memory (kB) context switches packets sent/received disk blocks read/written read/write transactions
Gray-box metrics (from the Spread or BFT protocols)	checkpoints (BFT) tokens received (Spread) membership changes (Spread) view changes (BFT) message retransmissions (both)

Table 1. Metrics collected by Gumshoe.

4.1 Instrumentation

We list the OS- and protocol-level metrics that Gumshoe collects in Table 1. We exploit the `sar` utility that is a part of the `sysstat` [3] performance monitoring tools for Linux to collect and log OS-level metrics from each replica’s node at one-second intervals. In addition, we instrument the two underlying fault-tolerant protocols, Spread and BFT, to monitor events that are indicative of the health of the system. These additional gray-box metrics are also shown in Table 1. We use the term *raw-metric traces* to represent the actual, unprocessed (hence, raw) time-series of the metrics gathered at each node.

5 Diagnosis Algorithms

We use system-wide peer-comparison to diagnose the culprit node, where *peer comparison* denotes the comparison of some numerical value across all the file-server replicas (the “peers”). As shown in Figure 1, we develop two variants of this approach to explore the tradeoffs between performing peer comparison on pre-processed data (the anomaly traces) vs. the unprocessed data (the raw traces):

- The *anomaly-trace* algorithm that first performs anomaly detection on each node’s raw metric-traces to generate a corresponding anomaly trace, followed by peer comparison on the resulting anomaly traces, and
- The *raw-trace* algorithm that performs peer comparison directly on the raw metric-traces.

Before describing Gumshoe in detail, we validate the hypotheses listed in Section 3, namely that (i) a performance problem will manifest on the culprit node’s metrics, and that (ii) it will cause the culprit node’s metrics to differ from the problem-free nodes’ corresponding metrics. Both hypotheses are borne out by our experimental observations of the two replicated file-systems.

As an example, Figure 2(a) shows the CPU usage and the change-point score¹ on 3 nodes of a BFS replicated file-system running the IOzone benchmark [2]. When we change the workload by increasing the number of clients from 1 to 2, we see correlated changes in the CPU usages of the 3 nodes. However, when we later inject a memory leak at a backup replica, only the backup replica’s node (the culprit) experiences a change in its CPU usage due to increased page swapping, while the CPU usages of the problem-free replicas remain similar and unperturbed.

This example also reinforces our peer-comparison approach. Peer comparison can discern the difference between the two cases—workload changes will manifest symmetrically (with possible time lags) in the steady-state on the metrics of all nodes, while a performance problem will manifest asymmetrically so that the culprit node’s metrics deviate from those of the problem-free nodes.

5.1 The raw-trace Algorithm

Gumshoe’s *raw-trace* algorithm identifies the culprit node by peer comparison of raw-metric traces using cross-correlation. At a high level, we compare the trends, rather than the absolute values, of a pair of input waveforms in order to establish their degree of similarity. In our case, we cross-correlate the traces for each metric across all of the file-server replica nodes, taking the nodes a pair at a time, after subtracting the mean value² from each respective node’s trace.

For each metric, we compute its similarity between each pair of nodes i and j using the Pearson correlation coefficient, ρ_{ij} , over a window, *correlationWin*, of samples of that metric. The coefficient $\rho_{ij} \in [-1, 1]$, where 1 indicates strong correlation, 0 indicates no correlation and -1 indicates strong negative correlation (one variable increases

¹A change-point score (a number between 0 and 1) indicates abrupt changes in the behavior of a time-series. The higher the change-point score, the more evident or the more dramatic the change.

²This allows us to focus on the comparison of the trends, rather than on the absolute values, of metrics.

while the other decreases) in that metric across nodes i and j . Due to our validated hypothesis (that problem-free nodes resemble each other in their metrics), $\rho_{ij} \simeq 1$ in problem-free conditions, even under workload changes. To identify culprit nodes, we use a threshold of 0.5 for ρ_{ij} to determine whether a metric is strongly correlated across two nodes; the documented acceptable range for a threshold is $\rho_{ij} \in [0.5, 1]$ [14].

Some metrics (e.g., memory usage) can remain relatively constant over certain windows of time, leading to $\rho_{ij} = 0$ over those windows. To address this, we introduce a threshold, *normalThresh*, to flag relatively constant metrics by assuming the metrics are maximally correlated if the difference between their standard deviations is less than *normalThresh*. We also used median filtering, a standard signal-processing technique, to remove any isolated outliers (impulse noise) from the data. Pre-processing data with median filtering allows us to ignore isolated instances of outliers, e.g., a single retransmission in a *correlationWin* that might otherwise result in an increased false-positive rate.

After computing all of the cross-correlations (i.e., for every metric for every pair of nodes), Gumshoe diagnoses the culprit to be the node that is:

[Rule 1] Anomalous in its metrics; or

[Rule 2] Anomalous in the most number of its metrics, if multiple nodes satisfy Rule 1; or

[Rule 3] Historically the most anomalous, i.e., the node that has exhibited the most anomalies in the previous *correlationWins*, if multiple nodes satisfy Rule 2.

5.2 The anomaly-trace Algorithm

We use singular spectrum transformation (SST) [16] to detect anomalies in each metric monitored on a node. At a high level, SST computes a summary of the performance metrics over fixed-sized windows in time and examines the differences in the summary data over multiple successive windows to see if the data has changed sufficiently. A change-point score is computed to quantify any abrupt change in the behavior of the summary data over the multiple successive windows. Change-point scores range from 0 to 1 and can be thought of as the probability that a change has occurred in the time-series, where 0 implies no discernible change, and 1 implies a discernible change.

To compute a summary of the data over a window, SST uses the Singular Value Decomposition (SVD) of the data over the window. The purpose of SVD is to extract the dominant features (called principal components) out of a very large dataset, to produce a smaller but sufficiently representative (hence, a summary) dataset that is more tractable to analyze. For each node, SST transforms the node’s raw-metric time-series into a new time-series of change-point

scores. SST computes the SVD over two overlapping windows (w), namely: (i) the past, which represents historical trends in the metrics, and (ii) the present, which represents the current trends in the metrics. SST then obtains the representative patterns over the windows by selecting the top r principal components. The change-point score, z , in a given window is the degree of change in the representative patterns across windows.

Thus, our anomaly detector requires us to tune two parameters, namely, the window size, w , and the number of principal components, r . A large w increases the false-negative rate because it leads to smoothing over a large window and can miss important trends in the data. On the other hand, a small w increases the false-positive rate because the change points are more susceptible to measurement noise. Ide and Tsudo [16] recommend setting r to 3 or 4.

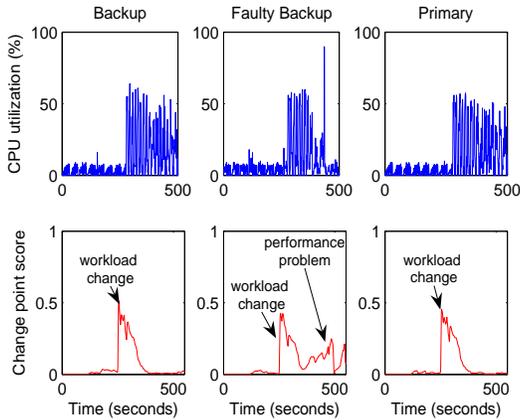
Our anomaly detection serves as a preparatory phase for the system-wide peer-comparison algorithm described in Section 5.1, with the raw-metric time-series replaced instead by the change-point time-series $\{z[1], z[2], z[3], \dots\}$. Our *anomaly-trace* algorithm allows us to investigate whether peer-comparison performs better when we pre-process the raw-metric traces into anomaly traces.

5.3 Online Problem Isolation

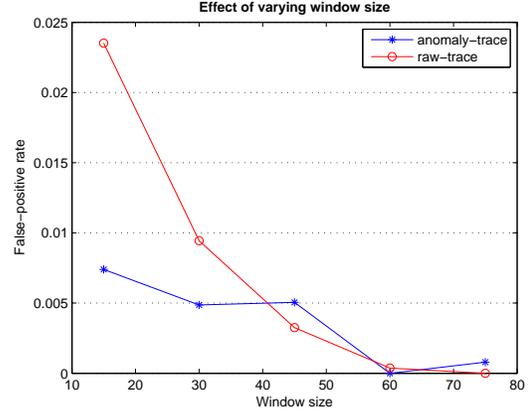
We describe how we enable problem isolation at runtime, by leveraging our diagnosis algorithms to discover the culprit node. For our runtime instrumentation, we use Ganglia [22], a scalable distributed monitoring system to collect the metrics listed in Table 1. Ganglia consists of two daemons: the Ganglia Monitoring Daemon *gmond* that we run on each node in our system to collect the performance metrics, and the Ganglia Meta Daemon (*gmetad*) that we run on a designated master node that periodically queries the *gmond* daemons for data, and records all the numeric, volatile metrics to a database.

By default, the *gmond* daemons collect only a subset of the metrics of interest to us. To expand the set of gathered metrics, we augment the Ganglia metric command-line tool, *gmetric*, with the *sar* utility mentioned in Section 4.1. Using *sar*, we collect OS-level metrics from each replica’s node at 5-second intervals and transmit these metrics to the *gmetric* tool, which sends them to the *gmond* daemons. In addition, we instrument the underlying fault-tolerant protocols, Spread and BFT, respectively to monitor protocol events that indicate the health of the system. We use *embeddedgmetric* [1], a package from Google code labs, to instrument the protocols and unicast metrics to the local *gmond* daemons. Our instrumentation framework imposed a 3% runtime overhead on file-system throughput.

We implemented the *raw-trace* algorithm to run online in the form of a Perl script at the dedicated diagno-



(a) A legitimate workload increase on BFS replicas causes correlated changes in CPU utilization, while a performance problem on the faulty backup manifests only on *that* node.



(b) False-positive rate of black-box diagnosis reduces with increase in anomaly-detection window size (w), for problem-free runs of replicated-CoreFS and BFS.

Figure 2. Effect of workload, performance problems, and anomaly-detection window.

sis server that hosts the Ganglia Meta Daemon (`gmetad`). The Perl script continually extracts metrics from Ganglia’s database, and then performs system-wide correlation of the raw-metric traces using a *correlationWin* of 30 samples at 5-second intervals, which yielded a low false-positive rate in our experiments. If our algorithm persistently flags a node as the culprit for more than half the duration of *correlationWin*, we proactively crash the culprit node to thwart it from degrading system performance further. This conservative strategy (of waiting for a culprit node to be flagged often enough before we indict it) minimizes the likelihood of spurious recovery actions, but might permit some problems to linger in the system while we wait to gather sufficient evidence to indict the culprit nodes.

6 Experimental Setup

Testbed. We conducted our evaluation on the Emulab distributed testbed [29] using 6 nodes (3000MHz processor, 2GB RAM, RedHat Linux kernel 2.4.18) connected by a 100Mbps LAN. In both the replicated-CoreFS and the BFS cases, we use a single client and 4 file-server replicas, each located on its own node. We used the default, out-of-the-box configuration for the underlying fault-tolerant protocols, i.e., 5 seconds for both Spread’s token-loss timeout and BFT’s view-change timeout.

Assumptions. We make the following assumptions in our experiments: (i) we inject only a single, independent problem at a time into one of the nodes hosting a file-server replica, (ii) clocks are synchronized (to within *correlationWin*/2 secs, i.e., 30 secs) across nodes in order to correlate the node-level metric traces for the purpose of peer com-

parison, and (iii) the nodes do not run any extraneous applications other than the target replicated file-system, the file-system benchmarks and the fault-tolerant protocols.

Workload. We generated the workload using two file-system benchmarks—Postmark [18] and IOzone [2]. Postmark is a random-access workload designed to emulate small file workloads such as e-mail. IOzone is a sequential access workload designed to emulate streaming workloads. We configured Postmark to use 10,000 files, 10,000 transactions, and 100 subdirectories. IOzone measured the performance of 64 KB sequential writes and reads to a single 100 MB file.

Problem injection. Exploiting the linker’s library interpositioning capability, we injected problems transparently into a single file-server replica by overriding specific system calls, in user space, as the replica executes. We explored the effect of a number of performance problems (listed in Table 2) that were modeled after those seen in real-world file-systems. We ran three sets of experiments for each injected problem per protocol and workload.

The problems described in Table 2 can be due to independent or common-mode faults. For example, message-loss due to a misconfigured Ethernet card could occur on all nodes, or be limited to a subset of nodes if configuration changes were incorrectly propagated. Gumshoe’s algorithms can localize problems provided that the majority of nodes exhibit correct behavior.

7 Experimental Evaluation

We evaluate Gumshoe by assessing the effectiveness—through the false-positive rate, problem-detection rate and

Table 2. Performance problems targeted in our experiments.

Problem	Real-world Incident	Mechanism for Inducing Problem
Resource exhaustion	User unable to delete OpenAFS volumes once server partition filled up [13]	Exhaust disk space by copying a large file to replica before the start of the experiment.
Abrupt crash	NFS mount daemon randomly crashes and panics the server [5]	Injected by abruptly killing the replica.
Periodic hang	Slow receivers trigger flow control mechanisms in multicast protocols [7].	Injected by intercepting the application’s <code>recv()</code> calls and putting the process to sleep for 100ms. Studies the effect of a slow receiver.
Message loss	Hub drops packets when bridging 100MB from server to client on 10MB segment [28]	Injected by intercepting the protocol’s <code>send()</code> and <code>recv()</code> calls to randomly drop messages at rates 20%. Studies the effect of message retransmissions and network partitions.
Large messages dropped	Lack of support for Ethernet jumbo frames at server results in dropping of large packets [6].	Injected by intercepting the protocol’s <code>send()</code> and <code>recv()</code> calls and dropping messages larger than 500 bytes. Studies the effect of blocking application-level messages and allowing protocol-level messages through.

precision —of its diagnosis algorithms.

We count each diagnosis towards one of these categories: (i) a true positive (tp), when a node is correctly identified to be the culprit; (ii) a false positive (fp), when a node is incorrectly identified to be the culprit; (iii) a false negative (fn) when we failed to detect the problem; (iv) a true negative (tn), when we correctly identify problem-free nodes. The *false-positive rate* is the fraction of innocent nodes that were wrongly flagged as problematic, i.e., false-positive rate = $fp / (fp + tn)$. The *problem-detection rate* refers to the fraction of problem-induced runs in which our algorithms identified the culprit. The *precision* refers to the fraction of culprit nodes that were correctly identified in problem-induced experimental runs, i.e., precision = $tp / (tp + fp)$.

By measuring the false-positive rate over multiple, repeated problem-free³ experimental runs and over a range of configurations, we tune Gumshoe’s parameters to achieve our goal of best isolating performance problems in replicated-CoreFS and BFS. As an example, we arrive at the appropriate setting for the anomaly-detection window size, w (described in Section 5.2), by measuring the false-positive rate over values of w ranging from 15 to 75, for problem-free Postmark workloads with both replicated-CoreFS and BFS, as shown in Figure 2(b). We observed low false-positives rates with $w = 60$.

Our experimental results are presented as follows. Section 7.1 compares the performance of Gumshoe’s `raw-trace` and `anomaly-trace` algorithms when only black-box metrics are used. Section 7.2 discusses whether gray-box metrics can improve diagnosis. Section 7.3 compares Gumshoe’s algorithms with the fault-tolerant protocol’s native failure-detection mechanisms, and Section 7.4 examines the effect of problem isolation on throughput.

³We use only the problem-free, and never the problem-induced, metric traces as training data for tuning Gumshoe’s parameters.

7.1 Black-box Diagnosis

Selection of metrics. For black-box diagnosis, we omitted those OS-level metrics in Table 1 whose cross-correlation was less than 0.5 in the problem-free traces. We found that network-related metrics (i.e., packets sent/received) worked best for both BFS and replicated-CoreFS.

Comparison of diagnosis algorithms. We evaluated the problem-detection rate and precision of Gumshoe’s black-box algorithms for both Postmark and IOzone (see Figure 3). The `raw-trace` algorithm exhibited the best performance at diagnosing the problems that we injected.

The `anomaly-trace` algorithm fared poorer than the `raw-trace` one because the anomaly-detector detects only the abrupt changes in metrics, and fails to detect the problem once the trends in the metrics settles down. For example, a memory leak that manifests as a gradual upward trend in memory usage might go unnoticed by the anomaly-detector. The `raw-trace` algorithm, on the other hand, would detect this anomaly if the memory usage at the problem-free replicas remained fairly constant while that of the culprit replica was constantly rising.

Effect of correlated problems in BFS. We expected Gumshoe to perform better with IOzone than with Postmark because the IOzone workload had less variation in behavior. However, contrary to our expectation, we experienced a 10% drop in precision with IOzone. We observed that the majority of false-positives occurred when we injected a message-loss problem at the primary replica, but Gumshoe pointed to a backup replica instead.

On further inspection, a correlated problem had resulted in the following way: (i) the primary’s message-loss problem led to increased message-retransmissions, causing the UDP buffer at a backup replica to overflow, leading to further message losses; (ii) this backup requested multiple view-changes and was excluded from the current view

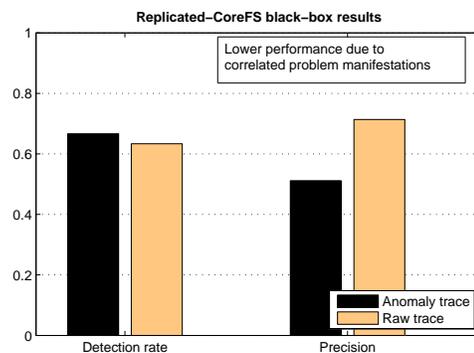
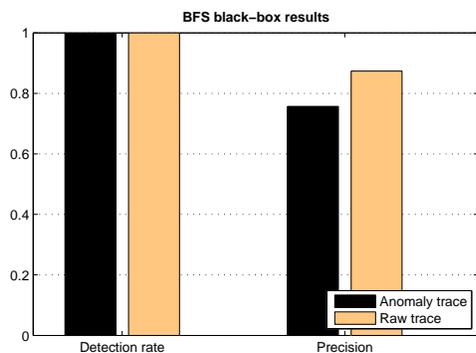


Figure 3. Performance of black-box algorithms. The raw-trace algorithm performed better than the anomaly-trace algorithm because the anomaly detector detects only abrupt changes in metrics, and fails to detect the problem once a trend in metrics is established.

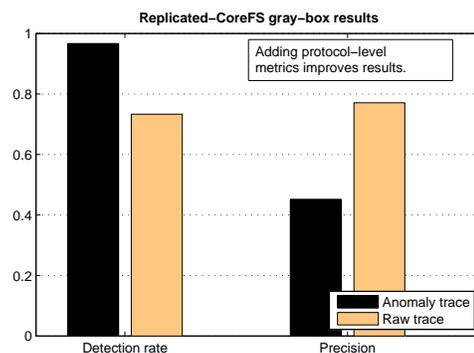
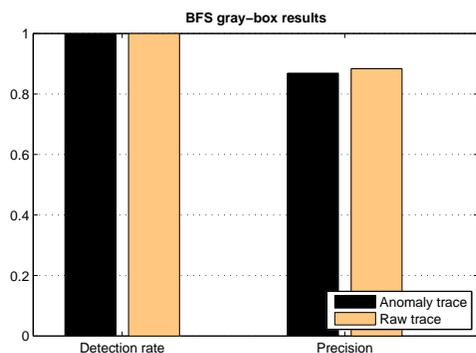


Figure 4. Performance of gray-box algorithms. The incorporation of gray-box (i.e., protocol-level) metrics improves the precision of diagnosing problems in Spread.

(Gumshoe correctly diagnosed this backup as the culprit). When we recomputed the problem-detection rate and precision to take into account the correlated problem, Gumshoe’s precision improved for IOzone to levels comparable to our Postmark experiments.

Effect of correlated problems in replicated-CoreFS.

Both algorithms were better at identifying the culprit node in BFS than in replicated-CoreFS. BFT, the protocol underlying BFS, is a quorum-based protocol that requires only a majority of server replicas to be correct in order to proceed. Thus, the sole culprit replica seldom retards the progress of the protocol and can be identified by its “odd-man-out” behavior as its metrics deviate from those of the other replicas. With replicated-CoreFS, performance problems (e.g., message loss) that hinder the progress of the protocol by slowing down the circulation of the underlying Spread protocol’s token result in correlated anomalous behavior being

exhibited by the metrics at all of the nodes in the system, making diagnosis difficult.

7.2 Gray-box Diagnosis

For replicated-CoreFS, gray-box (protocol-level) metrics include the number of received Spread protocol tokens and the number of retransmissions/second. For BFS, gray-box metrics include the number of checkpoints/second (we initially considered BFS’ message-retransmission rate, but this proved not to be well correlated across the server replicas). These metrics are indicative of the health of the system, and manifest with some uniformity across the server nodes in the system, making them ideal targets for peer-comparison. Gray-box metrics allow us to isolate problems that were previously indistinguishable with black-box metrics alone. For example, we can diagnose message-loss

problems in replicated-CoreFS by analyzing the differences in the message-retransmission rates across the replicas’ nodes. Figure 4 highlights the improvement in problem-detection rate and precision that occur through the consideration of gray-box metrics. For Replicated-CoreFS, the `anomaly-trace` algorithm’s ability to detect problems increases, albeit at the cost of decreased precision. The `raw-trace` algorithm, on the other hand, improves in both its problem-detection rate and precision.

7.3 Comparison to Protocols

As mentioned earlier, fault-tolerant protocols use timeouts to detect problems and evict nodes that the protocols regard as unresponsive. However, some problems can hide “under the radar” of the protocol’s timeouts and cause lingering performance problems to exist and even propagate within the system. Thus, we investigate opportunities for Gumshoe’s gray-box `raw-trace` algorithm to complement the protocols’ failure-detectors.

Gumshoe out-performs protocols. Gumshoe indeed localized certain performance problems ahead of the protocols, providing a window of opportunity for problem isolation. Gumshoe correctly diagnosed file-system problems, e.g., the disk-full problem, that were constrained to the file-system layer and did not impede the fault-tolerant protocol’s progress or its ability to deliver messages.

With a gray-box approach, Gumshoe was able to localize two other performance problems: 20% message-loss and dropping of large messages in replicated-CoreFS. The message-loss problem ultimately triggers the Spread protocol’s failure detectors, but the protocol isolates the culprit in only 1 out of 3 runs because the culprit node managed to respond within the protocol’s failure-detection timeouts and was repeatedly re-included in the new node-group membership. In the BFS case, the BFT protocols do not explicitly detect problems in the backup replicas, but Gumshoe can indeed diagnose such culprit backups, affording us an opportunity for problem mitigation.

Protocols out-perform Gumshoe. The protocols are better at detecting problems that abruptly halt their progress, e.g., abrupt crash in the replicated-CoreFS case and in the BFS primary replica, and a process-hang in the BFS primary replica. Both the replica-crash and the replica-hang problems highlight the difference between Spread’s daemon- and BFT’s library-based architectures.

With Spread, a node-specific daemon process represents the protocol and the file-server processes must connect to the daemon to avail of the protocol’s services. With BFT, a library represents the protocol, and file-server processes must be compiled with the library to avail of the protocol’s services. In Spread, a replica-hang does not affect the

progress of the underlying daemon because Spread disconnects the replica from its local daemon to avoid the protocol being hindered by a slow receiver. On the other hand, a replica-hang in the BFS primary replica provokes a view-change once the client detects the non-responsiveness of the primary. Here, the BFT protocols detect the problem in the primary ahead of Gumshoe.

7.4 Effectiveness of Problem Isolation

Figure 5 shows the effect of the 20% message-loss problem on the replicated-CoreFS filesystem running the Postmark benchmark, both with and without Gumshoe. The message-loss problem illustrates a scenario where the performance problem hid “under the radar” of Spread’s failure-detection timeouts, leading to a significant increase in response times when Gumshoe was deactivated. Gumshoe initiates problem isolation 204 seconds after the injection of the 20% message-loss problem. The effective elimination of the culprit node improves the read-throughput by 77%. On average, Gumshoe’s problem isolation increased throughput in replicated-CoreFS running the Postmark workload by 12%. In the BFS case, the improvement in throughput was 4%.

8 Experiences

Network-related problems can cause correlated performance degradations. Because fault-tolerant protocols involve network-intensive (message-passing) coordination, performance problems (e.g., message losses) that manifest solely on network-related metrics are difficult to localize using a black-box approach because these faults lead to correlated problem manifestations across the entire system due to the intrinsic coupling within the protocols. However, by leveraging protocol-level metrics, Gumshoe’s gray-box approach can identify the culprit node despite such correlated problem manifestations.

Diagnosis can complement the protocol’s native failure-detectors. Problem diagnosis techniques can complement the protocol’s native failure-detection mechanisms to provide for a more responsive strategy. Problem isolation is possible when sufficient system information exists to enable us to diagnose the culprit node ahead of the underlying protocols.

Metric selection. The network-related metrics at the OS-level, e.g., packets received per second, are good indicators of problems in both systems. The fault-tolerant protocols impose some regularity and coordination between the nodes, e.g., through the token’s circulation in Spread, and through the three-phase agreement protocol for each message sent in BFT. When choosing protocol-level metrics, a good rule-of-thumb is to seek metrics that manifest simi-

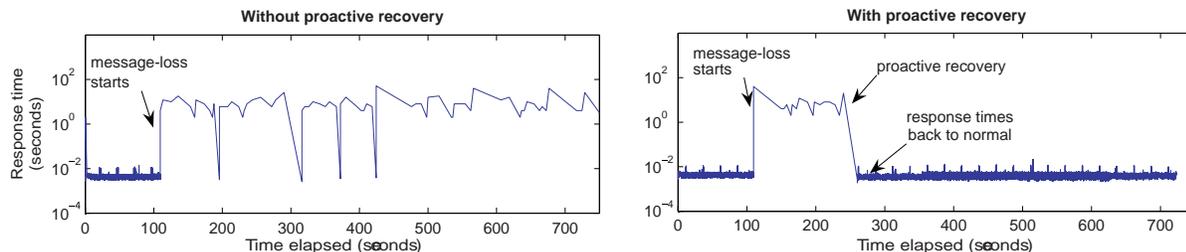


Figure 5. Problem isolation improves performance. The problem isolation action taken at 204 seconds improved the response time of the Postmark benchmark on replicated-CoreFS.

larly across replicas, under problem-free conditions (e.g., checkpoints in BFT, tokens received in Spread). Other reliable sources of information are metrics that will tend to manifest at a higher frequency at the culprit node, e.g., increased message-retransmission requests.

The fault-tolerant protocol’s architecture influences diagnosis. BFT requires a majority of the replicas to be correct in order to proceed. Therefore, the culprit replica seldom retards the progress of the protocol and can be identified by its “odd-man-out” behavior as it lags behind the other replicas in its processing of the client’s requests. In the Spread protocol, problems that hinder the progress of the protocol by retarding the circulation of the token (e.g., message loss) can result in correlated anomalous behavior at all of the nodes in the system, making diagnosis difficult.

With Spread’s daemon-based architecture, problems that manifest solely at the file-servers are better contained and lead to light-weight notifications. For example, the periodic-hang results in a simple socket-disconnection to evict the target server. With BFT’s library-based implementation, the file-servers are integrated with the protocols. Therefore, a file-server problem can cause both the server and its underlying protocols to be evicted.

9 Related Work

We have previously developed an error-reporting infrastructure for Emulab [17], to support future diagnosis of failures in Emulab. We have also applied machine-learning algorithms [23] to localize the source of correlated problem manifestations in replicated client-server systems, under static workloads. Gumshoe focuses instead on problem diagnosis in replicated file-systems through a novel peer-comparison algorithm that operates even under workload changes. We demonstrate Gumshoe conclusively by diagnosing real-world performance problems in replicated-CoreFS and BFS.

SSM [21], a system that manages user-session state in

distributed applications, is probably the closest to our research. SSM uses a statistical anomaly-detector that exploits peer comparison to detect problematic nodes, and reduce the sensitivity to workload changes. Gumshoe goes beyond SSM in highlighting the tradeoffs between approaches that use historical/local data vs. those using peer/global data.

Agile Store [20] is a file-system prototype that dynamically triggers variations in the system size and fault-threshold by statistically analyzing gray-box data to detect changes in the threat-level. Agile Store focuses on low-overhead approaches for tolerating arbitrary faults, whereas Gumshoe focuses on the early diagnosis and isolation of performance problems.

Proactive recovery through periodic reboot [8, 25, 10] provides a low-cost and effective technique for sustaining system availability in replicated systems. However, not all problems can be fixed by a periodic reboot. Gumshoe’s algorithms are focussed on isolating the culprit node. At present, the degree of replication in Gumshoe is fixed. Research by Sousa et al. [27], and Ramasamy and Schunter [24] provides guidance on how to vary design parameters to achieve desired reliability levels.

Farchi et al [15] describe a distributed debugger for detecting bugs such as deadlocks in DCS (the fault-tolerant protocol in Websphere). They test for local and global invariants in the protocol to detect problems. Gumshoe’s approach illustrates the feasibility of a black-box approach to problem isolation, and also shows how protocol-level metrics can improve precision. Cohen et al. [12] use black-box metrics to generate signatures of known problems that can be used for diagnosis. Gumshoe’s approach identifies the node that is the most likely cause of the problem and can be used without a historical database of problem signatures.

10 Conclusion

We have described the design, implementation and evaluation of Gumshoe, our diagnosis system that aims for

problem isolation in replicated file-systems. Gumshoe periodically gathers OS-level and protocol-level metrics, and then analyzes and cross-correlates these metrics across the replica nodes to automatically isolate the source of a performance problem. Through an empirical analysis of Gumshoe on two different replicated file-systems (replicated-CoreFS and BFS), two different underlying fault-tolerant protocols (Spread and Castro-Liskov BFT) and two file-system benchmarks (Postmark and IOzone), we show that Gumshoe’s diagnosis algorithms can indeed complement the fault-tolerant protocols’ native fault-detectors mechanisms.

References

- [1] Google code lab’s embedded gmetric. <http://code.google.com/p/embeddedgmetric/>.
- [2] The IOzone file system benchmark. <http://www.iozone.org/>.
- [3] Sysstat utilities. <http://pagesperso-orange.fr/sebastien.godard/>.
- [4] Y. Amir, C. Danilov, and J. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *International Conference on Dependable Systems and Networks*, pp 327–336, New York, NY, June 2000.
- [5] G. Barazer. [nfs] mountd randomly crash and panic the server. http://sourceforge.net/mailarchive/message.php?msg_name=461CFADA.1080305%40oxeva.fr, Apr. 2007.
- [6] H. Barth. [openafs-devel] delays and lost contact with fileserv with 1.3.84 and higher. <http://www.openafs.org/pipermail/openafs-devel/2005-October/013188.html>, Oct. 2005.
- [7] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.
- [8] G. Candea, A. Brown, A. Fox, and D. Patterson. Recovery-oriented computing: Building multiter dependability. *IEEE Computer*, 37(11):60–67, Nov. 2004.
- [9] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Symposium on Operating Systems Design and Implementation*, pp 173–186, New Orleans, LA, Feb. 1999.
- [10] M. Castro and B. Liskov. Proactive recovery in a Byzantine-fault-tolerant system. In *Symposium on Operating Systems Design and Implementation*, pp 273–288, San Diego, CA, Oct. 2000.
- [11] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):1–43, Dec. 2001.
- [12] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *Symposium on Operating Systems Principles*, pp 105–118, Brighton, UK, Oct. 2005.
- [13] S. Devine. [openafs] full disk woes. <https://lists.openafs.org/pipermail/openafs-info/2007-July/026718.html>, July 2007.
- [14] J. Devore. *Probability and Statistics for Engineering and the Sciences*. Duxbury Press, 2007.
- [15] E. Farchi, G. Kliot, Y. Krasny, A. Krits, and R. Vitenberg. Effective testing and debugging techniques for a group communication system. In *International Conference on Dependable Systems and Networks*, pp 686–699, Yokohama, Japan, Dec. 2005.
- [16] T. Ide and K. Tsuda. Change-point detection using Krylov subspace learning. In *International Conference on Data Mining*, Minneapolis, MN, Apr. 2007.
- [17] M. P. Kasick, P. Narasimhan, K. Atkinson, and J. Lepreau. Towards fingerprinting in the Emulab dynamic distributed system. In *USENIX Workshop on Real, Large Distributed Systems*, pp 7–7, Seattle, WA, Nov. 2006.
- [18] J. Katcher. Postmark: A new filesystem benchmark. Technical Report TR3022, Network Appliance, 1997.
- [19] V. Kher, M. Kokotovich, E. Seppanen, and Y. Kim. CoreFS network file system. <http://www.cs.umn.edu/research/sclab/coreFS.html>.
- [20] L. Kong, D. J. Manohar, M. Ahamad, A. Subbiah, M. Sun, and D. M. Blough. Agile store: Experience with quorum-based data replication techniques for adaptive byzantine fault tolerance. In *Symposium on Reliable Distributed Systems*, pp 143–154, Orlando, FL, Oct. 2005.
- [21] B. C. Ling, E. Kiciman, and A. Fox. Session state: Beyond soft state. In *Symposium on Networked Systems Design and Implementation*, pp 295–308, San Francisco, CA, Mar. 2004.
- [22] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7):817–840, July 2004.
- [23] S. Pertet, R. Gandhi, and P. Narasimhan. Fingerprinting correlated failures in replicated systems. In *USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, Cambridge, MA, Apr. 2007.
- [24] H. V. Ramasamy and M. Schunter. Architecting dependable systems using virtualization. In *Workshop on Architecting Dependable Systems*, Edinburgh, UK, June 2007.
- [25] H. P. Reiser and R. Kapitza. Hypervisor-based efficient proactive recovery. In *Symposium on Reliable Distributed Systems*, pp 83–92, Beijing, China, Oct. 2007.
- [26] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, Dec. 1990.
- [27] P. Sousa, N. F. Neves, P. Verissimo, and W. H. Sanders. Proactive resilience revisited: The delicate balance between resisting intrusions and remaining available. In *Symposium on Reliable Distributed Systems*, pp 71–82, Leeds, UK, 2006.
- [28] P. van Riezen. Re: [nfs] 2.2.18 problems with nfs 3. http://sourceforge.net/mailarchive/message.php?msg_id=Pine.SGI.4.30.0812142039490.39857-100000%40jones.lab.madscience.nl, Dec. 2000.
- [29] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Symposium on Operating Systems Design and Implementation*, pp 255–270, Boston, MA, Dec. 2002.