Fingerpointing Correlated Failures in Replicated Systems

Soila Pertet, Rajeev Gandhi and Priya Narasimhan
Electrical & Computer Engineering Department
Carnegie Mellon University, Pittsburgh, PA 15213-3890
spertet@ece.cmu.edu, rgandhi@ece.cmu.edu, priya@cs.cmu.edu

Abstract

Replicated systems are often hosted over underlying group communication protocols that provide totally ordered, reliable delivery of messages. In the face of a performance problem at a single node, these protocols can cause correlated performance degradations at even non-faulty nodes, leading to potential red herrings in failure diagnosis. We propose a fingerpointing approach that combines node-level (local) anomaly detection, followed by system-wide (global) fingerpointing. The local anomaly detection relies on threshold-based analyses of system metrics, while global fingerpointing is based on the hypothesis that the root-cause of the failure is the node with an "odd-man-out" view of the anomalies. We compare the results of applying three classifiers – a heuristic algorithm, an unsupervised learner (k-means clustering), and a supervised learner (k-nearest-neighbor) – to fingerpoint the faulty node.

1 Introduction

Distributed systems are vulnerable to the propagation of failures due to the inherent coupling between components. *Fingerpointing* (i.e., root-cause analysis, problem determination or failure diagnosis) is especially challenging in these environments because the resulting correlated failure manifestations can obscure the root-cause of the problem and can lead to potential red-herrings in diagnosis. We investigate the effectiveness of machine-learning techniques to fingerpoint correlated performance problems in distributed replicated systems.

Replication is commonly used for providing fault-tolerance to distributed client-server applications. Replicated systems often exploit group communication protocols [5] for the totally ordered, reliable delivery of all messages to and from the replicated server. Group communication protocols use timeouts to detect failures, and attempt to reduce all failures to group-membership changes, i.e., a slow node, a lossy network all ultimately trigger a membership change. However, some failures can hide "under the radar" of the protocol's timeouts and cause performance problems to linger and even propagate within the system.

Our approach to fingerpointing correlated performance problems combines *node-level (local) anomaly detection* with subsequent *system-wide (global) fingerpointing*. Local anomaly detection relies on threshold-based analyses of system metrics. Global fingerpointing is based on the premise that the root-cause of the failure is likely the node with an "odd-man-out" view of the anomalies. For instance, the manifestation of the failure might be most severe at the faulty node, or the faulty node might have a different view of the anomalies, e.g., the faulty node displays a surge in one metric while the other nodes display a dip in the same metric.

We perform our investigations in the context of state-machine replicated servers [11] hosted on top of two different group communication protocols, namely, the Spread token-ring protocol [2] and Castro-Liskov BFT [4]). We inject faults at a single (server replica) node, and monitor various system metrics in a black/gray-box manner at the faulty and non-faulty nodes in the system. We compare the results of applying three classifiers – a heuristic algorithm [9], an unsupervised learner (*k-means clustering*), and a supervised learner (*k-nearest-neighbor*) – on the gathered system metrics, to fingerpoint the faulty node.

Our initial results show that the performance of the classifier depends on the extent to which the local anomaly detectors capture the asymmetric behavior between the faulty node and the non-faulty nodes. For example, all classifiers performed well at fingerpointing the process hang and the memory leak. However, because group communication protocols involve network-intensive coordination, faults (such as packet losses) that affect network traffic were difficult to diagnose using a black/gray-box approach alone because they led to correlated failure manifestations across the entire system.

This paper is organized as follows: Section 2 and 3 motivate and present our fingerpointing approach; Section 4 and 5 discuss our experimental configuration and empirical observations; Section 6 discusses related work, and Section 7 outlines our conclusions.

2 Motivating Example

Token-ring group communication protocols, e.g., Spread, impose a *logical* ring on the set of nodes constituting the node-group membership. A special message called the token circulates within the node-

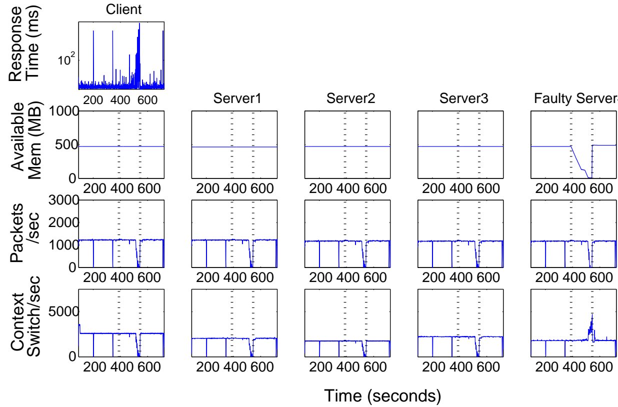

Figure 1: A memory leak was injected in Spread-hosted server4 at ∼400 seconds. This causes correlated performance degradations on the non-faulty servers and results in increased response times at the client. The faulty server eventually crashes at ∼600 seconds.

group, sequentially from one node to the next. A node is allowed to broadcast messages to the other nodes only when it holds the token. This circulation of the token around the ring is critical to achieving consensus on message ordering and group membership. Since nodes can only broadcast messages when they hold the token, a performance slowdown due to a faulty node can manifest even at non-faulty nodes, leading to correlated performance degradations.

Figure 1 shows the propagation of failure manifestations in a system with 4 server replicas and a single client hosted on top of Spread. A memory leak is injected at ∼400 seconds at `server4`, where available memory is first metric to exhibit an anomaly. The memory leak eventually slows down `server4` due to increased paging activity as `server4` runs out of memory. The memory leak in `server4` results in a slowdown in the token's circulation, with resulting drops in network traffic and context-switch rates at even the non-faulty nodes. The client observes increased response times, although the client is simply selecting the first response that it receives from any of the server replicas. `server4` finally crashes at ∼600 seconds, and is subsequently restarted. This example illustrates two challenges in fingerpointing, namely:

► **Failure manifestation changes "shape":** Initially, the memory leak manifests only as a drop in available memory. However, as `server4` runs out of memory, the corresponding effect manifests on other metrics, such as network traffic and context-switch rate.

► **Failure manifestation propagates to non-faulty nodes:** Due to the inherent coupling in the system, a performance slowdown on one node results in a correlated slowdown on non-faulty nodes, thereby obscuring the root-cause of the problem.

3 Fingerpointing Approach

Because correlated failure-manifestations can arise on multiple nodes in the system, our approach combines local (node-level) anomaly-detection with global (system-wide) fingerpointing. We examine the differences in the various nodes' view of anomalies rather than comparing the nodes' raw metric values because each node might have a different (raw-metric) view of what is normal.

3.1 Instrumentation Framework

We instrumented each server node in the system to collect time-series data of the application-, OS- and protocol-level metrics at runtime. Because our current fingerpointing granularity is the node, we focus on node-level metrics and do not consider process-level metrics. We collected OS-level metrics by sampling the `/proc` pseudo-filesystem every second. In addition, we monitored network traffic using the `libpcap` packet-capture facility. We used these network traces to generate logs of aggregate network traffic in terms of packets/sec.

We instrumented Spread to keep track of the number of tokens received per second, the number of message-retransmissions per second, and membership changes. For BFT, we monitored the checkpoint frequency, the message-retransmission rate and membership changes. The BFT checkpointing frequency was set to approximately once every 2.5 seconds. We converted the checkpoint event-series data to time-series data of checkpoints/second by averaging checkpoints over 20-second intervals. At the application level, we monitored the response time at the client-side of the application.

3.2 Local Anomaly Detection

We used a simple statistical approach to detect anomalies in the performance metrics collected on each node. We used fault-free training data to compute initial estimates of the mean (μ) and the standard deviation (σ) of each performance metric. We then computed an adaptive-μ and adaptive-σ for each metric as a weighted combination (λ=0.95) of the previous estimate of the mean and the current observation. The adaptive algorithm helped us deal with slow changes in operating conditions.

We flagged anomalies if the performance metric's value fell beyond $\pm 6\sigma$ threshold from the adaptive-μ.

Table 1: Metrics collected.

OS-level	Available memory (bytes)
	Context switches/second
	Packets/second
Protocol-level (Spread)	Tokens received/second
	Message retransmissions/second
Protocol-level (BFT)	Checkpoints/second
	Message retransmissions/second
App-level metrics	Response time

For each sample period, we generated an anomaly vector showing the anomalous state of the performance metric. A "1" in the anomaly vector indicated an anomalous metric, while "0" indicated no anomaly. We also generated anomaly vectors using multiple thresholds based on the representation proposed in [8]. Anomalies were characterized as extremely low (-6σ), low (-3σ), normal, high ($+3\sigma$) and extremely high ($+6\sigma$). These five states were respectively represented as integers ranging from -2 to 2.

The only metric that was the exception to this rule was available memory, where we opted for a μ-based threshold. Memory usage was fairly constant and the σ-based threshold resulted in a high false-positive rate. We used a threshold of $\pm$ 0.5% and $\pm$ 0.25% of the mean, μ, instead of $\pm 6\sigma$ and $\pm 3\sigma$ respectively, to detect anomalies.

As an example, if `server4` had a memory leak, the node-level anomaly vectors might resemble the following, for the metrics:

[memory, packets/sec, context-switches]

server4: [-2, 0, 0]; server3: [0, 0, 0]; server2: [0, 0, 0]

These vectors indicate that `server4` experienced an extremely high (thus, the -2) anomalous memory behavior, while the other server nodes experienced no anomalies (thus, the 0) in any of their metrics.

Noise filtering We used a two-phase process to filter out any "noise" and to construct a perfect anomaly detector with a zero false-positive rate. In the first phase, we required that an anomaly be detected in about 50% of the metric's observed values in the window of length *anomalyWin* before logging it. In the second phase, we trigger fingerpointing only if anomalies are logged in more than 50% of the samples in a window of length *fingerpointWin* of any metric. We tuned our anomaly detector to yield a zero false-positive rate by setting *fingerpointWin*=15, and logging anomalies only if we observed 3 or more anomalous points in a window of length *anomalyWin*=7.

We initially planned to trigger fingerpointing when the client-side response time violated desired service-level objectives (SLO). However, some faults, e.g., process hangs, were masked by the server's replication and did not adversely impact client-side response times.

3.3 Global Fingerpointing

The anomaly-detection process in Section 3.2 served as a preparatory phase for our fingerpointing algorithm. Due to the inherent coupling in group communication protocols, we fingerpoint the faulty node by comparing deviations from normal behavior across the nodes that host server replicas in the system, instead of focusing on the behavior of only a single node. To perform fingerpointing across the server nodes, we synchronized the generated anomaly logs using timestamps.

We investigated three approaches to fingerpointing namely, a heuristic approach, unsupervised clustering (k-means), and supervised clustering (k-nearest neighbor).

3.3.1 Heuristic Approach

The heuristic fingerpointer [9] examined the anomaly-vector logs of the performance metrics across all of the nodes, in each *fingerpointWin*, using the following rules. (i) If some node is markedly the only one to be problematic in one or more of its metrics, then, we fingerpoint that particular node. (ii) If more than one node is problematic in one or more of its metrics, then, we fingerpoint the node that is problematic in the most number of its metrics. (iii) If more than one node is problematic in the most number of metrics, then, we fingerpoint the node (if one exists) that has historically exhibited anomalies in previous *fingerpointWin*s. This fingerpointer examines the anomaly-vector logs produced using the single $\pm 6\sigma$ threshold described in Section 3.2.

3.3.2 k-Means Clustering (Unsupervised)

We used MATLAB's implementation of k-means clustering for our unsupervised learning algorithm. This fingerpointer is based on the premise that, during the period of performance degradation, the system exhibits two dominant types of behavior: (i) the failure as perceived by the faulty node, and (ii) the failure as perceived by the non-faulty nodes. Therefore, we set $k = 2$.

As with the heuristic approach, the k-means fingerpointer examined the anomaly-vector logs of the performance metrics across all of the nodes, in each *fingerpointWin*. We investigated the effectiveness of using anomaly-vector logs with a single threshold (denoted by `kmeans`) and with multiple thresholds or severity levels (denoted by `kmeans+sev`). We used the sum of absolute differences, (i.e., L1) distance-measure because the anomaly vectors use normalized values, rather than the raw values, of the metrics.

For fingerpointing, we assume that the majority of the nodes are fault-free, that they have a similar view of the failure and can, therefore, be grouped in the larger cluster. We assume that the faulty node will be grouped into the smaller cluster, and that the faulty node will be the most frequently occurring node in this smaller cluster. If we fingerpoint more than one node in a *fingerpointWin*, we examine the nodes that we historically fingerpointed in previous *fingerpointWin*s. If only one node was fingerpointed historically, we flag that node as faulty, otherwise we flag all of the fingerpointed nodes in the current window as faulty (i.e, we have an obscured diagnosis).

To reduce the possibility that `kmeans` converges to a local minimum, we repeat the clustering process 10 times in each *fingerpointWin*, starting with randomly selected centroids for each repetition. We then choose the solu-

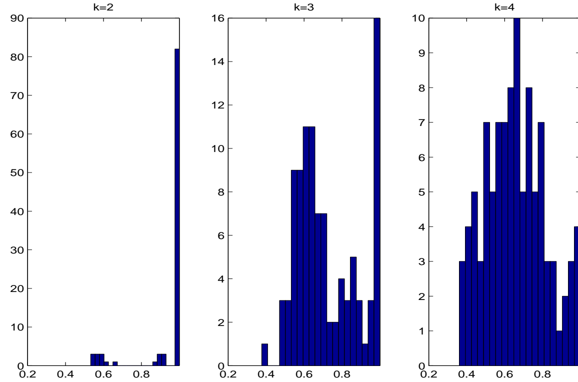

Figure 2: Correlation similarity measures for increasing values of k for memory leak in Spread. For $k = 2$, similarities are concentrated about 1 (most solutions highly similar). For $k > 2$, the wide spectrum of similarities implies there is no longer one correct solution.

tion that minimizes the intra-cluster distance.

Cluster stability. One of the main challenges in unsupervised clustering is selecting the optimal number of clusters. We investigated our hypothesis that there are only two dominant clusters in the data by using the correlation similarity measure proposed in [3]. This similarity measure ranges from 0 to 1, with values closer to 1 indicating high similarity. The idea is that if the similarity between clusterings obtained by taking multiple random subsamples of the data is high, then, the partitions found by clustering are meaningful. However, if the similarity measures are more widely distributed, this implies that there is no preferred solution and that the same data point may be classified into different clusters depending on the random selection of the initial centroids.

Figure 2 shows the distribution of similarity measures for a 15-second *fingerpointWin* for the memory leak in Spread. This window contains anomaly vectors with multiple severity-levels for OS-level metrics generated when the failure manifested on multiple nodes in the system. We varied the cluster sizes from 2 to 4 and used subsamples of 80% of the dataset. We computed similarity measures for 100 random subsamples of the data for each cluster size. For $k = 2$, the similarities are concentrated around 1 (i.e., most solutions are highly similar). For $k > 2$, we observe a wide spectrum of similarities, implying that there is no longer one correct solution. Our hypothesis held for memory leaks and process hangs, but was violated for packet-loss faults due to the symmetric manifestation of the fault across all nodes in the system.

Changing failure manifestations. The clustering algorithm also experiences instability when the failure manifestation changes, thereby violating the assumption that $k = 2$. For example, when the memory leak progresses from manifesting only on available memory at the faulty node to manifesting as drops in network-traffic and context-switches across all nodes. We mitigated this instability by utilizing a temporal sliding window of length (*fingerpointWin*) so that the changing failure manifestation is typically limited to a single window.

3.3.3 k-Nearest Neighbor (Supervised)

We used MATLAB's implementation of the k-nearest neighbor (denoted by `knn+sev`) for our supervised learner. `knn+sev` classifies objects based on the closest training examples in the feature space. We set $k = 1$ as this minimized misclassifications in our data. As with the k-means clustering, we used the L1 distance-measure. We provided training examples using anomaly vectors with multiple severity levels for each type of fault in Spread and BFT. Anomaly vectors were labelled as either fault-free or faulty. With `knn+sev`, we could distinguish between different types of faults (e.g., we could diagnose that `server4` has a memory leak, as opposed to the "blanket" diagnosis that `server4` is faulty). However, for the sake of comparison against the other two fingerpointing techniques, we used the coarser labels.

The disadvantage of structuring the anomaly vectors as described in Section 3.2 is that the correlated failure manifestations cause some anomaly vectors for different faults to appear the same. For example, both the memory leak at non-faulty Spread nodes and the packet-loss fault at a faulty Spread node manifest as drops in network traffic and context switches. These ambiguous anomaly vectors can lead to misclassification.

A work-around for this might be to use a single aggregate anomaly vector that represents anomalies across all the nodes. For the sample anomaly vectors shown in Section 3.2, we would now obtain an aggregate anomaly vector for a memory leak in `server4` to be:

[-2,0,0, 0,0,0, 0,0,0]

(interpret as [<**server4**>, <**server3**>, <**server2**>]).

Similarly, a memory leak in `server3` would be:

[0,0,0, -2,0,0, 0,0,0]

However, this approach would need require more training examples, one for each kind of fault at every node in the system.

We therefore opted to use the first approach, despite its potential for misclassification, primarily to provide a fair comparison against the other two fingerpointers. Once we trained the `knn+sev` fingerpointer, we used it to classify each anomaly vector in the *fingerpointWin* as either faulty or fault-free. If a majority of the anomaly vectors in *fingerpointWin* are classified as faulty, we fingerpoint that node. As with the k-means fingerpointer, if we fingerpointed more than one node in a *fingerpointWin*, we looked at the nodes that we had historically fingerpointed in the previous *fingerpointWin*s. If only one node was fingerpointed in the previous *fingerpointWin*s, we

flagged this node as faulty, otherwise we flagged all of the fingerpointed nodes in the current window as faulty.

4 Data Collection

We conducted our experiments in the Emulab distributed testbed [13] using 5 nodes (850MHz processor, 256kB cache, 512MB RAM, RedHat Linux kernel 2.4.18) connected by a 100Mbps LAN. In both the Spread- and the BFT-supported configurations, we used a simple state machine-replicated client-server test application, with one client and four server replicas, each on its own node. The client sent a 1024-byte request to the server at 10ms intervals. In addition, we used the default membership timeouts (5 seconds) for both Spread and BFT.

Each experiment covered 30,000 round-trip client requests and ran for ∼10 minutes. We collected traces for the metrics listed in Section 3.1, and injected the 3 faults identified in Section 4.1. We ran each experiment 3 times, yielding a total of 27 runs (i.e., (Spread + BFT Leader + BFT Follower) × (3 faulty) runs × 3 times).

4.1 Fault Injection

Exploiting the dynamic linker's library interpositioning capability (through the `LD_PRELOAD` environment variable in Linux), we implemented an interceptor to inject faults transparently into a process by overriding specific system calls, in user space, as the process executes. Our fault injection target was either the server replica or the protocol running on a designated node in the system. We injected the following performance-degrading faults:

▶ Memory leak: We injected a memory leak by bypassing the replica's `free()` system call. To accelerate the rate of the leak, we modified our server to allocate/deallocate 96kB with each request, as a part of normal operation. This fault studied the effect of gradually loading a node and starving the protocols of memory.

▶ Process hang: We intercepted the replica's `read()` system call and blocked the replica for several minutes. This fault investigated the effect of a slow receiver on the protocol's flow control.

▶ Packet-loss fault: We intercepted the protocol's `send()` and `recv()` calls and randomly dropped incoming and outgoing packets at packet-loss rates of 20% of the packets at the node. This fault investigated the effect of message retransmissions and network partitions in the protocol.

5 Results of Fingerpointing

At the end of each run, each fingerpointing technique diagnosed a set of nodes as faulty. We categorized each node in this set as either: (i) a true positive (`tp`), i.e., we correctly fingerpointed the guilty node.; (ii) a false positive (`fp`), i.e., we incorrectly fingerpointed an innocent node.; (iii) a false negative (`fn`), i.e., we failed to fingerpoint the guilty node.

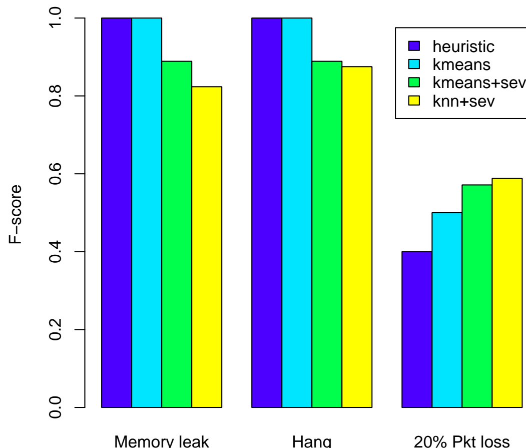

Figure 3: Performance of the three fingerpointing techniques across various faults.

We then calculated `precision` and `recall`. `Precision` refers to the fraction of nodes fingerpointed that were indeed faulty, i.e., `tp/(tp+fp)`. Whereas `recall` refers to the probability that a node is fingerpointed given that it is faulty, i.e., `tp/(tp+fn)`.The `F-score` is the harmonic mean of precision and recall. The `F-score` ranges from 0 to 1; an `F-score` equal to 1 implies perfect diagnosis.

5.1 Preliminary Analysis

Figure 3 shows the results of our fingerpointing techniques. The `F-score` is reported for each type of fault across all runs for the Spread- and BFT-supported configurations. Each technique performed well at fingerpointing the memory leak and process hang. Memory leaks and process hangs manifested asymmetrically on the faulty and non-faulty nodes, for example, memory leaks manifested as severe drops in available memory on the faulty node. These asymmetries were highlighted by the local anomaly detectors, facillitating global fingerpointing.

The heuristic fingerpointer and the unsupervised single-threshold `kmeans` perfectly diagnosed the memory leak and process hang. The k-means algorithm with multiple thresholds/severity levels (`kmeans+sev`) and the supervised k-nearest neighbor performed slightly worse. The performance of `kmeans+sev` was degraded when the fault changed its manifestation – at these transition points, the anomaly vectors did not group into two clusters alone because the fingerpointing window contained multiple faulty behaviors. The supervised k-nearest neighbor clustering, `knn+sev`, on the other hand, performed worse when the correlated manifesta-

tion caused some anomaly vectors for different types of faults to look the same, resulting in misclassification.

All techniques performed poorly at diagnosing the packet-loss faults because these tended to manifest symmetrically across all nodes as drops in traffic. However, the use of multiple thresholds/severity levels in `kmeans+sev` and `knn+sev` improved diagnosis because the local anomaly detectors were better able to highlight subtle differences between the nodes. For example, highlighting drops in checkpointing frequency in BFT, and in the case of `knn+sev`, capturing a slight increase in message retransmission in Spread.

6 Related Work

Current research in application-level root-cause analysis centers on identifying the faulty components along the causal request path [1, 6]. However, components along the causal request path whose behavior is identified as anomalous may not always be the source of the problem. This may occur due to hidden dependencies between nodes that are not directly related to the request call-graph. Our approach provides insight on how to diagnose such failures in distributed, replicated systems.

Pip [10] helps programmers find bugs in distributed systems by comparing the actual system behavior against the expected behavior. Pip can identify performance problems in paths outside the user's causal request path. Pip requires programmers to annotate their systems with expectations of normal behavior whereas we profile system metrics to build templates of normal behavior.

Pinpoint [7] and PeerPressure [12] both use peer comparison to diagnose problems. Pinpoint diagnoses partial failures in J2EE environments whereas PeerPressure diagnoses configuration errors. Our approach also uses peer comparison to diagnose performance problems which can propagate through the system, thereby obscuring the root-cause of the problem.

7 Conclusion

Combining *node-level (local) anomaly detection* with subsequent *system-wide (global) fingerpointing* can aid in fingerpointing correlated failures in replicated systems. The accuracy/performance of fingerpointing depends on how good the local anomaly detectors are at highlighting any asymmetric behavior between the faulty node and the non-faulty nodes. For example, faults which manifest asymmetrically such as memory leaks and process hangs are easier to diagnose than packet-loss faults, which tend to manifest symmetrically across all nodes as drops in traffic. The use of multiple thresholds/severity levels improved the diagnosis of packet-loss faults because the local anomaly detectors were better able to highlight subtle differences between the nodes.

Overall, it appears that machine-learning techniques can assist in fingerpointing some types of correlated performance problems in distributed systems. Propagating and morphing failure manifestations are likely to warrant additional research into the combination of multiple machine-learning techniques in order to fingerpoint the faulty node with high accuracy and low false-positive rates, in both replicated and heterogeneous distributed systems.

References

[1] AGUILERA, M. K., MOGUL, J. C., WIENER, J. L., REYNOLDS, P., AND MUTHITACHAROEN, A. Performance debugging for distributed systems of black boxes. In *Symposium on Operating Systems Principles* (Boston Landing, NY, October 2003), pp. 74–89.

[2] AMIR, Y., DANILOV, C., AND STANTON, J. A low latency, loss tolerant architecture and protocol for wide area group communication. In *International Conference on Dependable Systems and Networks* (New York, NY, June 2000), pp. 327–336.

[3] BEN-HUR, A., ELISSEEFF, A., AND GUYON, I. A stability based method for discovering structure in clustered data. In *Pacific Symposium on Biocomputing* (Lihue, Hawaii, January 2002), pp. 6–17.

[4] CASTRO, M., AND LISKOV, B. Practical Byzantine fault tolerance. In *Symposium on Operating Systems Design and Implementation* (New Orleans, USA, February 1999), pp. 173–186.

[5] CHOCKLER, G. V., KEIDAR, I., AND VITENBERG, R. Group communication specifications: A comprehensive study. *ACM Computing Surveys 33*, 4 (December 2001), 1–43.

[6] COHEN, I., ZHANG, S., GOLDSZMIDT, M., SYMONS, J., KELLY, T., AND FOX, A. Capturing, indexing, clustering, and retrieving system history. In *Symposium on Operating Systems Principles* (New York, NY, USA, October 2005), pp. 105–118.

[7] KICIMAN, E., AND FOX, A. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks: Special Issue on Adaptive Learning Systems in Communication Networks 16*, 5 (September 2005), 1027– 1041.

[8] MAXION, R., AND FEATHER, F. A case study of ethernet anomalies in a distributed computing environment. *IEEE Transactions on Reliability 39*, 4 (October 1990).

[9] PERTET, S., GANDHI, R., AND NARASIMHAN, P. Group communication: Helping or obscuring failure diagnosis? Tech. Rep. CMU-PDL-06-107, Carnegie Mellon University Parallel Data Lab Technical Report, June 2006.

[10] REYNOLDS, P., KILLIAN, C., WIENER, J. L., MOGUL, J. C., SHAH, M. A., AND VAHDAT, A. Pip: Detecting the unexpected in distributed systems. In *Symposium on Networked Systems Design and Implementation* (San Jose, CA, May 2006), pp. 115–128.

[11] SCHNEIDER, F. B. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR) 22*, 4 (December 1990), 299–319.

[12] WANG, H. J., PLATT, J. C., CHEN, Y., ZHANG, R., AND WANG, Y.-M. Automatic misconfiguration troubleshooting with PeerPressure. In *USENIX Symposium on Operating Systems Design and Implementation* (San Francisco, CA, December 2004), pp. 245–258.

[13] WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, S., NEWBOLD, M., HIBLER, M., BARB, C., AND JOGLEKAR, A. An integrated experimental environment for distributed systems and networks. In *Symposium on Operating Systems Design and Implementation* (Boston, MA, December 2002), pp. 255–270.