Tiresias: Black-Box Failure Prediction in Distributed Systems

Andrew W. Williams, Soila M. Pertet and Priya Narasimhan
Electrical & Computer Engineering Department
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3890

Abstract

Faults in distributed systems can result in errors that manifest in several ways, potentially even in parts of the system that are not collocated with the root cause. These manifestations often appear as deviations (or "errors") in performance metrics. By transparently gathering, and then identifying escalating anomalous behavior in, various node-level and system-level performance metrics, the Tiresias system makes black-box failure-prediction possible. Through the trend analysis of performance metrics, Tiresias provides a window of opportunity (look-ahead time) for system recovery prior to impending crash failures. We empirically validate the heuristic rules of the Tiresias system by analyzing fault-free and faulty performance data from a replicated middleware-based system.

1. Introduction

Current approaches to making distributed applications and systems fault-tolerant rely on first detecting a failure before initiating any form of fault-recovery action. With this kind of reactive (or "after-the-fact") fault-recovery, the impact of the fault is not necessarily averted. This approach also fails to take advantage of any pre-failure indicators or symptoms that might be present in the system. While analyzing these indicators might not avert the fault, the results of this analysis might have allowed recovery to be initiated faster or proactively, thereby mitigating the impact of the fault on the application/system.

Pre-failure indicators might occur in the form of a fault's manifestation on the behavior of various performance metrics in the system. For example, a distributed client-server, or middleware, application typically relies on network communication in order to dispatch invocations and responses, and to function correctly. Thus, the network becomes a critical resource to such applications. In such a middleware application, a memory-leak fault at the server can cause the server to slow down sufficiently as to degrade the server's responsiveness from the client's perspective. The network traffic between the client and the server will be seen to drop, perhaps even to the extent that the client receives no timely responses from the server. This might trigger the expiration of application- or middleware-level timeouts, which, in turn, raises a client-side exception that effectively indicates the server has failed. Note that the network itself has not failed in this case – rather, the application's behavior, from a network-traffic viewpoint, has changed to the extent that it is no longer representative of the normal, non-faulty execution of the application. Thus, the observation of the memory-leak's manifestation on network traffic might have provided an early indication of subsequent application-level failure.

Our approach is based on the hypotheses that *a fault manifests as increasingly unstable performance-related behavior before escalating into a failure*, and that systems exhibit steady-state performance behavior with few variations in the non-faulty case. Thus, the early discovery of any deviations, or anomalies, in the trends of various performance metrics can provide sufficient *look-ahead time* (a window of opportunity ahead of impending system failure), where we can perform proactive recovery.

The hypothesis of steady-state behavior to enable anomaly detection has been reported in the literature and also observed by us [1]. The hypothesis of increasing pre-failure instability is borne out by the literature, where a number of interesting computer-related faults have been observed to be preceded by a visible pattern of abnormal behavior that favors the use of some form of prediction. Typically, these failures result from gradual degradation faults, such as resource exhaustion [6], or from transient and intermittent hardware faults, such as disk crashes [8] or telecommunication equipment failures [16]. We acknowledge that

1-4244-0910-1/07/$20.00 ©2007 IEEE.

This work has been partially supported by the NSF CAREER grant CCR-0238381, the Army Research Office grants DAAD19-01-1-0646 and DAAD19-02-1-0389.

some failures might occur so abruptly that we cannot possibly hope to predict them. For example, if someone accidentally unplugs the power supply of a node, it might not be possible to predict the power outage simply because there was no discernible "build-up" to the failure.

The Tiresias system monitors various performance metrics (e.g., network traffic, CPU usage, context-switch rate), transparently to the application. By using the gathered time-series data to develop *a priori* models of normal, non-faulty application performance, and then examining these metrics under faulty conditions, Tiresias is able to generate predictions of impending failure without any knowledge of the application's internals. In this paper, we describe the results of the empirical study of Tiresias on a fault-tolerant middleware test-bed, where we inject various faults, such as memory leaks, thread leaks, etc. Our objective is to demonstrate the feasibility of black-box failure prediction through the analysis of performance metrics in a distributed system.

For Tiresias' performance-centric failure-prediction, we leverage heuristics that were originally developed for predicting single-node hardware failures. These heuristics were driven off data collected from analyzing error-logs [8]. We extend these heuristics and apply them quite effectively, for the first time, to predicting application-level failures in a *distributed* setting. Unfortunately, because applications are not normally instrumented to produce error logs of performance deviations, Tiresias generates its own "error logs" through the application of anomaly-detection techniques on the performance-metric data that it gathers. Equipped with these "performance-error logs" extracted from the time-series data of various performance metrics, we apply our extended heuristics to forecast application-level failure. The key contributions of this paper are as follows.

Performance metrics facilitate black-box failure prediction: Comparing the trends in the behavior of various performance metrics, under non-faulty and faulty conditions, can provide advance notice (look-ahead time) of application-level failures. Thus, Tiresias forms the first stage of a proactive fault-recovery framework. No application-level knowledge is needed to perform the monitoring or the subsequent failure prediction.

Empirical validation of Tiresias: We perform our experiments on a fault-tolerant middleware test-bed implementation. We inject a variety of faults into this test-bed, and discuss how Tiresias can be tuned to improve the quality of its predictions.

The remainder of this paper is organized as follows. Section 2 discusses the challenges and the assumptions underlying our approach. Section 3 provides the details of the Tiresias system. In Section 4, we describe our data collection and our experimental results. We discuss future and related work in Sections 5 and 6 respectively and we conclude in Section 7.

2. System Model

The Tiresias system makes some assumptions in order to assert statements about the future state of the distributed system. For one, faults that lead to failures, regardless of the fault's root cause (e.g., an application-level problem, such as a memory leak, or a network-level fault, such as a network-interface card failure), are assumed to affect performance metrics in an observable, identifiable way. These effects should be discernible as anomalous behavior when compared to normal system performance. This assumption is not far-fetched – in fact, it is fundamental to many anomaly-detection techniques [4, 5].

We also assume that failures, while appearing to be random to an outside observer, can exhibit performance patterns leading up to the failures. These *pre-failure patterns* can take the form of anomalous performance behavior prior to the failure. This assumption is borne out by other research: "predictable faults produce a transient performance degradation before causing a full-blown failure" [15], in the context of network faults. We regard escalating/cumulatively anomalous behavior (i.e., increasing departure from non-faulty behavior) to be indicative of impending failure.

We assume that our anomaly detector has sufficient resolution (i.e., sufficient granularity of time over which the anomaly-detection algorithm works) to generate a performance-error log that serves as input to Tiresias' failure prediction. In this paper, our smallest sampling interval of measurement is seconds (e.g., we measure network traffic in terms of packets per second). This is an artifact of our chosen anomaly-detection method (we emphasize that our intent in this paper is not to develop a new anomaly detector, but rather to exploit an existing, off-the-shelf anomaly-detection algorithm). Clearly, an anomaly detector that uses a higher resolution could predict deviations in performance metrics in a shorter window of time.

Tiresias cannot predict any failure that is not preceded by escalating anomalous behavior of the system's performance metrics. Application-level faults (e.g., value faults where the application produces a wrong result) that do not impact any performance metrics will go undetected. In addition, if our anomaly detector does not adequately capture the pre-failure symptoms, or if it has a high false-positive or false-negative rate, then, our predictive capability will clearly suffer.

3. Tiresias' Failure-Prediction Framework

As shown in Figure 1, we use a two-stage approach to analyzing our data: anomaly-detection followed by failure-prediction. In the first stage, we examine the performance-metric data to determine where it deviates from its expected

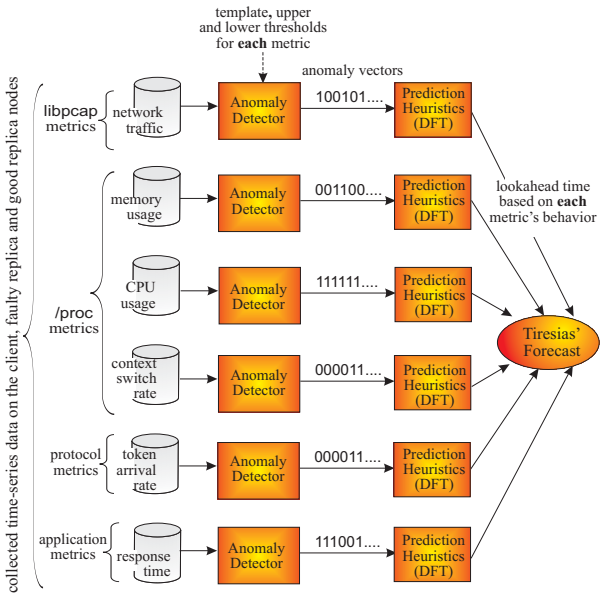

Figure 1. Stages in the operation of Tiresias.

value, thereby signaling an anomaly (i.e., "statistically unusual performance" [3]) that might be indicative of a potential problem in the system.

Tiresias learns of the normal state of the distributed system through the trends of the various performance metrics that it gathers. Note that the system could be multi-modal, e.g., there could be off-peak and peak workloads on the system. In this case, Tiresias builds multi-modal templates of the system's performance behavior. This allows Tiresias to keep up with the system's evolving behavior and to modify its conclusions according to workload changes.

In this anomaly-detection phase, we process all of the time-series data of the performance metrics using a relatively straightforward threshold-based anomaly detector [3] to determine when performance degradations occur. For each performance metric, the anomaly detector outputs an anomaly vector that indicates whether each data-point of the metric's time-series is deemed anomalous or normal. Basically, each metric's anomaly vector is a time-series consisting of a binary sequence of 1s and 0s to indicate the anomalous or normal state of that metric over time. The anomaly vector captures the state of the metric while allowing us to reduce the raw metric values into a simple binary representation.

We then apply the Dispersion Frame Technique (DFT) prediction heuristics [8] to the anomaly-vectors that we generate in the anomaly-detection phase. The DFT stage examines the anomaly vector for each metric, looking for specific clustering patterns of anomaly points. We exploit the DFT heuristic rules to seek patterns of behavior that might indicate escalating instability in a metric's behavior. If we observe such patterns of behavior, we fire warnings of impending failure. The rules do not distinguish between different kinds of failure and do not have access to application or fault-type information. Thus, the predictions can be useful to system administrators as advance indicators of system failure.

We note that the predictions do not reveal the root cause and do not guarantee that the failure will occur; they merely indicate escalating instability in the specific metric under examination. We also note that our system does not depend on the anomaly detection algorithm. Any anomaly detector would serve our purpose equally. The point of this paper is not to showcase either the anomaly-detection algorithm or the failure-prediction heuristics, nor even to highlight the combined power of both. Instead, the idea is to study Tiresias' synergistic usage of these existing techniques, and to evaluate the feasibility of its black-box failure-prediction strategy that is driven off performance metrics alone.

3.1 Data Collection

We identify which performance metrics are affected by a failure, and how they are affected, through collecting empirical data from our test-bed. We collected this data under non-faulty experimental runs, as well as under the injection of a variety of faults.

We monitored node-level resource usage on every node in our system by retrieving the resource-usage statistics from the `/proc` pseudo-filesystem [2] every five seconds. The `/proc` pseudo-filesystem contains a hierarchy of special files that represent the current state of every running process on the Linux operating system. By periodically examining `/proc/meminfo` and `/proc/stat`, we obtained information about the combined resource usage of all of the running processes on that node. We could opt to examine individual process-level resource behavior in addition to node-level aggregate behavior – however, for the purposes of this paper, the granularity of failure prediction is the node, and thus, the node-level aggregate performance metrics suffice for our purposes. Tiresias monitors the following system metrics for every node through that node's `/proc` filesystem.

- CPU usage (%) - The percentage of time that the CPU on the node is busy executing user and kernel tasks.
- Available memory (bytes) - This is the sum of both the free and the cached memory on a node. Cached memory is the amount of memory that Linux uses for the disk cache, and can be replaced quickly if a running (or new) program needs memory.
- Context-switch rate (per second) - This represents the number of context switches that the node undergoes in one second.

Tiresias monitors the network traffic and records a timestamp every time that a new packet appears on the wire. To monitor the network traffic in packets/sec, Tiresias uses a network sniffer based on the `libpcap` library [12] developed at Lawrence Berkeley National Laboratory. This library is widely used in intrusion-detection schemes and in packet sniffers. This library provides a high-level interface to capture all network traffic into and out of a node. In promiscuous mode, all of the packets on the network, even those destined for other nodes, are accessible through this mechanism.

On each node in the network, Tiresias monitors the rate of all traffic to and from that node. Tiresias records a timestamp every time that a new packet appears on the wire. Each host's packet-timestamp log is then locally analyzed to count how many packets have been seen in any given time interval. This script generates a network-load log that consists of the magnitude of network traffic (in packets/sec) for all of the 1440 minutes during a given 24-hour day. This network-load log that Tiresias generates is then used by its anomaly detector.

Our network-traffic capture is entirely passive on the network, and does not add any overhead to the existing network traffic. We also note that both the `/proc` and the `libpcap` capture mechanisms are transparent to the application.

3.2 Anomaly Detection

We emphasize that the Tiresias system can use *any* anomaly-detection scheme. The novelty of our work does not lie in the anomaly-detection scheme, but rather, in its exploitation for the prediction of failures.

Our current anomaly detector is based on algorithms that were developed and tested for network-related failures [3]. Simply computing the mean of the data and looking at a three-standard-deviations ($\pm 3\sigma$) rule for finding anomalies has been shown to be insufficient for metrics such as network traffic. These performance metrics are inherently dynamic, i.e., they vary with the workload, the time of day, the day of the week, etc. Thus, a true anomaly detector should account for the time-varying nature of the system's performance.

We employ a standard method [3, 9] for finding anomalous points. Using this method, we develop a *template*, a model of time-varying expected/normal system behavior for each metric, and *envelopes*, which represent tolerance limits for that metric's normal behavior. We describe the anomaly detector's analysis of network-traffic data; the anomaly detector is similarly applied to all of the other metrics.

We define four vectors X, P, T and V. The X vector is the current raw network-traffic data as captured by our sniffer. P is the smoothed traffic data, where smoothing maintains the general trend of the data, but avoids extreme discontinuities that could skew the data. T is the template that describes normal network-traffic behavior. This is the aggregate of the smoothed data that is combined into a single expected/normal-behavior pattern. Finally, V is a variance template that is used to calculate the standard deviation that we use as the thresholds for detecting anomalies. This represents the range of values that we define as upper and lower bounds for the acceptable, non-anomalous network traffic. The following four steps, are then used to detect anomalous network-traffic data points.

- Compare X both to T and to the threshold values calculated from V. Any points outside the envelope are considered anomalous, and are flagged in the corresponding anomaly vector.
- Smooth the raw data to produce a one-day trend, P.
- Combine the current prediction template with P to produce the new template, T_1. This is done by using exponential smoothing.
- Compute the current variance vector, V, to include the new data.

We implemented a simple, but adequate, median and mean smoother. In median or mean smoothing, a point is replaced with the median or mean, respectively, of the surrounding N points. Large values of N will produce smoother curves, but can effectively smooth out important outliers in the data. The new template T_i is found by

$$T_i = (1 - \alpha)\, T_{i-1} + \alpha P_{i-1} \quad i \geq 0$$

where α weights one data set more than another. This controls how quickly new data is incorporated into the template T. If T_0 does not exist, then, P_0 becomes T_1. The formula for finding the variance vector, V_i, is

$$V_i = (1 - \alpha)\, V_{i-1} + \alpha\, (P_{i-1} - T_{i-1})^2 \quad i \geq 0$$

The choice of α, $\alpha \geq 0$,determines how quickly the system folds changes into the templates [3]. The current network data, X, is compared to the thresholds, which are calculated to be three standard deviations (3σ) above and below the template T; using $\pm 3\sigma$-based thresholds is a common statistical technique. Any values of the real-time network data that fall outside these thresholds are deemed anomalous and are appropriately recorded in our performance-error log.

The template, the upper and lower thresholds are all derived from multiple traces for each metric over multiple, independent non-faulty experimental runs of the entire system. Armed with this information, Tiresias can generate an anomaly vector for any supplied input trace for that metric. Thus, the end-result of this stage is a collection of anomaly vectors, one for each metric on every node in our system.

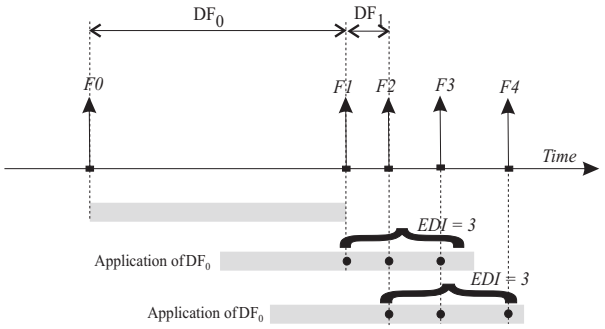

Figure 2. Illustration of the DFT Rule 3.3.

3.3 Dispersion Frame Technique (DFT)

The Dispersion Frame Technique (DFT) [8] was originally developed to analyze and predict failures in hardware devices, such as disk drives. DFT ascertains the relationship between anomaly occurrences by examining how closely they occur in time. It consists of a set of heuristic rules developed from extensive empirical studies of error-logs of disk drives and memory boards. These rules have only been applied to single-node hardware failures and to readily available error event-logs as they currently exist on operating systems. Tiresias represents the first instance of applying these rules to the derived time-series error-logs (i.e., the anomaly vectors or performance-error logs) that are output by our anomaly detector.

DFT utilizes the concept of Dispersion Frames (DF) and Error Dispersion Indices (EDI). A Dispersion Frame is the inter-arrival time between successive error events, i.e., successive anomalies of the same metric. The EDI is defined as the number of errors (or anomaly occurrences) in one half of a DF. A highly related group of anomalies, which is a sign of instability in the system and a potential harbinger of failure, exhibits a high EDI. In Figure 2, we show a series of errors (F_1,F_2,..), with DFs (DF_0,DF_1,..) being centered on them. Given three consecutive errors (e.g. F_1, F_2 and F_3 for DF_0) in the latter half of a DF, the EDI is equal to 3.

In the original DFT implementation, a DF of 168 hours was used to activate the heuristics. This number was based on the statistical analysis of data gathered from kernel error-logs and actual hard-drive failures for the Andrew filesystem at Carnegie Mellon. Five heuristic clustering rules were developed and were used when the DF fell below the activator of 168 hours.

- 3.3 rule: when two consecutive indices from successive applications of the same DF exhibits an EDI of at least 3, as illustrated in Figure 2;
- 2.2 rule: when two consecutive indices from two successive DFs exhibits an EDI of at least 2;
- 2-in-1 rule: when a DF is less than one hour;
- 4-in-1 rule: when four error events occur within a 24-hour frame;
- 4-decreasing rule: when there are four monotonically decreasing DFs and at least one frame is half the size of its previous DF.

The reasoning behind each rule can be found in more detail in [8]. DFT was successfully applied to predicting hard-drive failures on the Andrew File System at Carnegie Mellon University, where it achieved a 93.7% success rate [7].

3.4 Reconfiguring DFT for the Tiresias System

Tiresias aims to investigate the effectiveness of DFT, an established single-node hardware-failure prediction technique, for failure prediction in distributed systems.

Tiresias analyzes the generated anomaly vectors with its reconfigured DFT rule-set. Its DFT-like heuristics are activated when the time interval between two successive anomalies drops below a predetermined frame size, which we call the minimum activator. The unique nature of each system forces us to find a minimum activator for it. Tuning the activator can affect the look-ahead time, as we describe later. The minimum activator can be determined either by an experienced administrator, or by examining multiple fault-free and faulty system traces and then estimating the appropriate activator.

After the right minimum activator is configured, Tiresias begins analyzing the anomaly vectors generated in the previous phase. Once the heuristics are activated, a number of clustering rules are applied to the anomaly vectors in order to fire warnings of imminent failure. The key idea here is to find the EDI as we progressively scan each anomaly vector and as the DF changes. Tiresias' prediction rule-set differs from the original clustering rules described in [8], and has been modified to work effectively with our fault-tolerant middleware:

- 3.3 rule: when two consecutive indices from the successive application of the same DF exhibit an EDI of at least 3.
- 2.2 rule: when two consecutive indices from two successive DFs exhibit an EDI of at least 2.
- 4-decreasing rule: when there are four monotonically decreasing DFs and at least one DF is half the size of its previous one.
- 2-in-10 rule: when a DF is less than 10 minutes.

We could elect to fire a failure prediction when any one of these rules is valid. However, we can perform a further clustering of the rules in time by recording when each rule is fired. For instance, we could fire predictions only when we have seen x firings of one rule and y firings of another rule. Subsequent firings of the rules signifies a trend of increasing rule firings, where each rule firing in itself represents a trend of increasing performance anomalies. Thus, the accumulation of multiple rule firings allows the Tiresias system to increase its confidence in predictions about impending failures in the system.

4. Empirical Evaluation

We evaluated Tiresias on a fault-tolerant middleware system running on the Emulab test-bed environment.

4.1 Test-Bed Description

Our empirical evaluation was conducted in the Emulab distributed environment [17]. In our experiments, we used up to 4 physical nodes (850MHz processor, 256 Kb cache, 512MB RAM, running RedHat Linux kernel 2.4.18). The nodes were interconnected by a 100Mbps LAN. We use a simple two-tier distributed client-server test application, with one client and a dual-redundant server, with the client and the two replicas each on its own node. The client sends a request to the server, in response to which the server returns 32 Kb of data to the client; the client pauses 10ms between requests.

We use the off-the-shelf open-source MEAD fault-tolerant middleware [11] to provide replication support to distributed client-server applications. MEAD implements active (where all replicas are peers and actively executing) and passive replication (where one replica is the primary and the others are backups) styles using group communication protocols. This leads to some similarities in failure prediction for the two replication styles.

4.2 Data Collection

Each experiment covers 45,000 round-trip client invocations and runs for about 15 minutes. We collect traces for every one of our 5 metrics of interest (CPU usage, free memory, context-switches/sec packets/sec, token-arrivals/sec) on every node in the system, and in addition, the response time on the client node. Thus, we have 15 metrics altogether across the 3 nodes, plus the additional response-time metric on the client side, giving us a total of 16 metrics altogether. In addition, we perform this experiment for two different replication styles. Thus, we have a total of 32 traces of metrics that cover both the replication styles. We perform the collection of multiple traces (320,

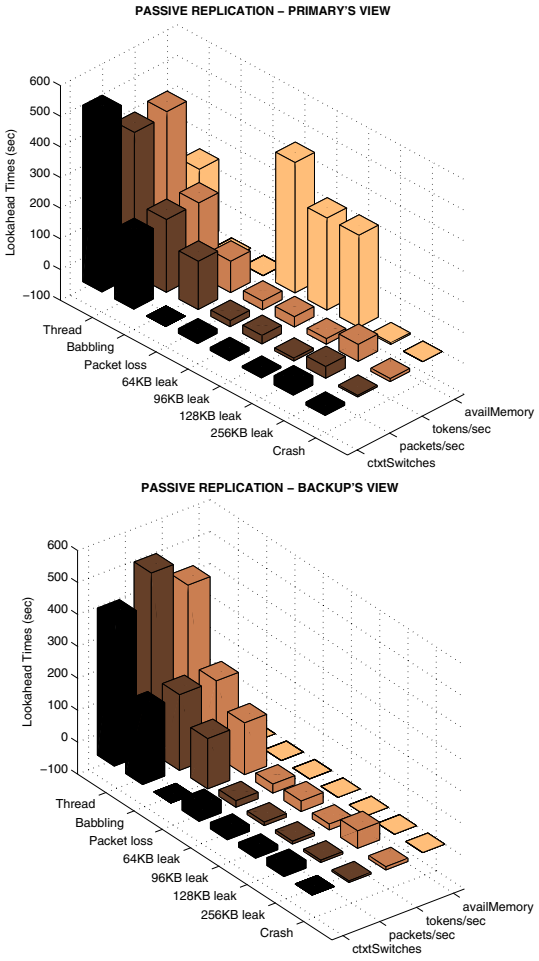

Figure 3. Look-ahead times for passive replication from the (i) faulty primary node's viewpoint and (ii) non-faulty backup node's viewpoint.

to be precise) of these 32 metrics in the fault-free case in order to provide a better basis for defining our normal template for anomaly detection. In addition, we inject 12 different performance-degrading faults (memory leaks of 64Kb, 96Kb, 128Kb, 256Kb at the primary replica and then at the backup, thread-leak, babbling node, packet loss, abrupt crash), which leads to 384 metric traces for studying failure prediction in our system.

4.3 Observations

Regardless of the replication style, Tiresias' failure prediction can predict the onset of various kinds of failures through its monitoring of system metrics, as shown in Figures 3 and 4. The amount of look-ahead time depends on the replication style, the failure, and the metric being moni-

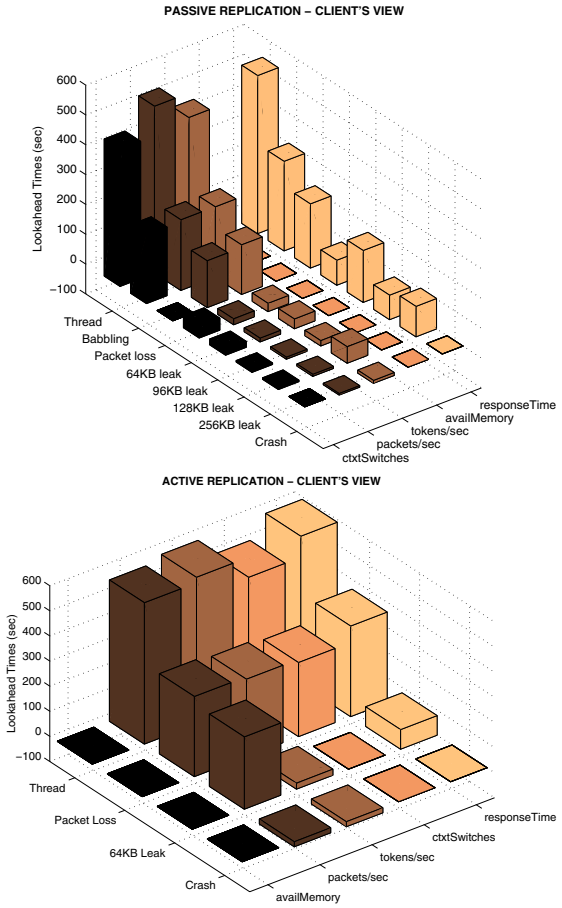

Figure 4. Look-ahead times from non-faulty client node's viewpoint for (i) passive replication and (ii) active replication.

tored. It is important to point out that our graphs show that look-ahead time (i.e., failure prediction) can be performed on a node other than the faulty one!

It appears that response time is the best harbinger of failure because it provides the most look-ahead time. Unfortunately, CPU usage was not stable enough for consistent failure-prediction and we were forced to resort to other metrics for this purpose. Also, in the case of the faults where we varied the failure rate, we noticed that the failure rate does affect the look-ahead time – the trend seems to be that the faster the failure, the smaller the look-ahead time, which makes intuitive sense. Certain kinds of failures (that show no period of instability prior to the failure) cannot be predicted, e.g., abrupt crashes.

We note that our failure predictions did encounter false positives. We recognized the false positives because we recorded when the faults were injected into the system. The false anomalies (and ,therefore, false predictions) were registered prior to the fault injection. We note that we had only 5 false positives out of 200 runs of our prediction algorithm (2.5%). We note that we used only simple $\pm 3\sigma$ anomaly-detection and also used the first rule-firing in DFT to predict failures. By using clustering and other $\pm n\sigma$ thresholds, we expect to lower this false-positive rate.

We also examined the effect of tuning the DFT failure-prediction algorithm with respect to its minimum activator (window) size. As a first observation, the look-ahead time for the 64Kb leak was greater than that for the 256Kb leak. This makes sense because the 256Kb leak caused the system to fail faster in comparison to the 64Kb leak.

As the DF activator size increases, more anomalous points are included in the DFT rulesets. These points, referred to as the window size, acts as a threshold for discriminating between anomalous points that are related and those that are simply transient hiccups. As more points are considered to be related, there is a better chance that some combination of points will match a DFT rule. At some point, the window is large enough to encompass a set of anomalous points that will always fire a DFT rule (e.g., 10 anomalies in a row). As the window size becomes larger, this set of points will always fire a prediction. This also increases the false-positive rate of our failure prediction. If these points are early on in the experiment, or other anomalous points do not appear ahead them, then, the look-ahead time is constant no matter how large the window.

5. Future Work

Our future work focuses on analyzing additional characteristics of Tiresias and exploring its extensions to root-cause analysis and proactive recovery.

As a part of understanding Tiresias' mechanisms better, we would like to inject other kinds of faults outside of those that we have studied in this paper. We would also like to investigate Tiresias' false-positive rate more accurately by gathering more empirical data. Another direction to pursue is to employ alternative anomaly-detection algorithms, along with DFT, within the Tiresias system to see if the accuracy and sensitivity of the predictions can be tuned better.

We also intend to develop a Distributed DFT (DDFT) algorithm within Tiresias. It is our belief that correlating failure predictions for various metrics across the system could be more revealing than predictions based on single metrics alone.

6. Related Work

The area of fault management and prediction remains an open field of research. Many approaches to fault detection and prediction have been proposed. Hajii et al. [4] have

developed a system based on the stochastic approximation of the maximum likelihood function. They propose a baseline of normal network operations with anomalous points defined as a sudden jump in the mean of the multivariate random variable. These jumps represent system anomalies and are used to detect network faults or performance problems.

Hood and Ji [5] propose a proactive fault-detection system that extracts anomalous points and analyzes them with a Bayesian network. This system is capable of detecting previously unseen faults. This system also uses anomalous network points as an indicator of network faults. The Bayesian network is also capable of filtering out some of the false positives, giving a more reliable fault detector.

Thottan and Ji [13] propose a system that analyzes statistics from the standard Management Information Base (MIB) variables [10] that represent the state of the network. The MIB data is analyzed using a sequential Generalized Likelihood Ratio test. This test indicates anomalous changes in the MIB variables. Unusual changes can represent an anomaly in the state of the network. The anomalies are used to detect network faults. Thottan and Ji [14] extended their previous system to propose a system similar in concept to ours. Their approach uses their previous MIB fault detector. The enhanced system uses intelligent agents to monitor the MIB data from the routers and provide time- and space-correlated alarms. The time-correlated changes in the individual MIB variables are spatially correlated using a combining scheme. Their system works on the assumption that the MIB variables will behave abnormally prior to, and during, a network fault. The detected anomalous points are used to predict the onset of network faults and degradations. Our system uses a similar concept, but instead of MIB variables, we use simple packets/sec statistics. Our system could be implemented to exploit the MIB anomaly detector in [13].

7. Conclusion

By transparently gathering, and then identifying escalating anomalous behavior in, various node-level and system-level performance metrics, the Tiresias system makes black-box failure-prediction possible. It exploits clustering rules that were originally developed for single-node hardware failures, and extends them successfully to predict, with reasonably confidence, application-level failures in a distributed system. In our experiments with injecting a variety of different faults in a reliable middleware system, the Tiresias system's predictions came in advance of the failures, thereby paving the way for proactive fault-recovery and high availability.

References

[1] T. A. Dumitraş and P. Narasimhan. Fault tolerance and the magical 1%. In *ACM/IFIP Conference on Middleware*, pages 431–441, Grenoble, France, November 2005.

[2] R. Faulkner and R. Gomes. The process file system and process model in UNIX System V. In *Winter USENIX Conference*, pages 243–252, Dallas, TX, Jan. 1991.

[3] F. Feather. *Fault Detection in an Ethernet via Anomaly Detectors*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1992.

[4] H. Hajii, B. Far, and J. Cheng. Detection of network faults and performance problems. In *Internet Conference*, Osaka, Japan, November 2001.

[5] C. Hood and C. Ji. Proactive network fault detection. *IEEE Transactions of Reliability*, 46(3):1147–1156, 1997.

[6] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton. Software rejuvenation: analysis, module and applications. In *International Symposium on Fault-Tolerant Computing*, pages 381–390, Washington,DC, June 1995.

[7] T.-T. Y. Lin. *Design and Evaluation of an On-Line Predictive Diagnostic System*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1988.

[8] T.-T. Y. Lin and D. P. Siewiorek. Error log analysis: Statistical modeling and heuristic trend analysis. *IEEE Transactions on Reliability*, 39(4):419–432, Oct. 1990.

[9] R. Maxion. Unanticipated behavior as a cue for system-level behavior. In *International Phoenix Conference on Computers and Communications*, pages 4–8, Phoenix, AZ, March 1989.

[10] K. McCloghrie and M. Rose. Management information base for network management of TCP/IP-based Internets:MIB-2. *RFC 1213*, 1991.

[11] P. Narasimhan, T. A. Dumitraş, S. M. Pertet, C. F. Reverte, J. G. Slember, and D. Srivastava. MEAD: Support for real-time fault-tolerant CORBA. *Concurrency and Computation: Practice and Experience*, 17(12):1527–1545, 2005.

[12] TCPDump. www.tcpdump.org.

[13] M. Thottan and C.Ji. Adaptive thresholding for proactive network detection. In *IEEE International Workshop on Systems Management*, pages 108–116, Newport, RI, April 1998.

[14] M. Thottan and C. Ji. Fault prediction at the network layer using intelligent agents. In *IEEE/IFIP Integrated Network Management*, pages 745–759, Boston, MA, May 1999.

[15] M. Thottan and C. Ji. Properties of network faults. In *IEEE/IFIP Networks Operations and Management Symposium*, pages 941–942, Honolulu, HI, 2000.

[16] R. Vilalta and Ma Sheng. Predicting rare events in temporal domain. In *IEEE International Conference on Data Mining*, pages 474– 481, Maebashi, Japan, December 2002.

[17] 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*, pages 255–270, Boston, MA, December 2002.