Architecture-Driven Diagnosis of Performance Failures in a Token Ring

Andrew Williams and Priya Narasimhan
Electrical & Computer Engineering Department
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3891
andrewwi@andrew.cmu.edu, priya@cs.cmu.edu

Abstract

Communication infrastructures that provide distributed systems with key services can also end up being the medium whereby faults propagate through the system. We have previously observed that a single faulty node can degrade the performance of other, non-faulty nodes in the system. We present a method for identifying the node that is the origin of the failure by examining the architecture-driven constrained network-flows in a distributed system. By identifying the effects of the failure on the network, combined with our knowledge of the network-flow constraints, we can trace the effects of the failure back to its source node. We empirically evaluate our methods on a data set that was generated by injecting multiple performance-faults into a replicated middleware system with an underlying token-ring based group communication protocol. We correctly identify the faulty node in the case of failures that significantly change the performance characteristics of the network.

1. Introduction

Distributed systems rely on communication infrastructures to provide clients with key services. However, these infrastructures are also the medium through which faults can propagate from a faulty node, impacting the system as a whole and causing failures in the system [8]. Quickly locating and isolating the source of the failure can reduce recovery time, expense, system administrator workload, and keep the overall impact of the failure on the system to a minimum. This problem is complicated by the distributed nature and interconnectivity of the systems themselves. Many interrelated and interconnected components often interact in complicated and unpredictable ways, while different failures can manifest in diverse ways at different nodes in the system. Fingerpointing (also known as problem determination or failure diagnosis) consists of identifying a failure in the system and tracing the source of the failure to the culprit. In a distributed system, a failure can typically only propagate through the communication infrastructure. Assuming that the failure does not violate the group communication protocol, the failure can only propagate along the *constrained network* path.

1.1 Problem Statement

In this paper, we look at isolating the source of a performance fault in a distributed, replicated system using our knowledge of the constrained network-flows and system behavior. This paper aims to answer the following question: Knowing the constrained network flows and given that a failure has occurred, can we isolate the source of the failure by monitoring simple network metrics?

1.2 Approach

In order to address our research question, we study a replicated distributed system. The system runs over a token-ring group communication protocol, called Spread [2], to maintain message ordering and reliability. The token-ring approach of Spread greatly constrains the network flow at any given point in time. We inject a variety of faults into the running system and use simple fault detectors on each node in the system. Using our knowledge of the network flows, failure detection, and overall system time (synchronized across nodes), we locate the source of the failure in the system.

In order to detect the failures, we use a simple anomaly detector placed at each node. The detectors monitor a number of performance metrics and examines them for anomalies. The detectors are trained on normal operations of the system and then used to determine when the monitored metric is three standard deviations away the "normal".

1.3 Contributions

We show that the source of the fault can be identified using architecture-driven network constraints and simple net-

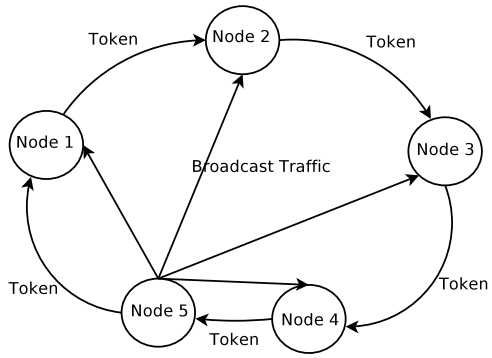

Figure 1. Five-member node group, showing the point-to-point token traversal and broadcast messages from node 5.

work metrics. The group communication protocol is both the medium through which failures propagate, as well as the origin of the network constraints that allow us to identify the root-cause of the failure. The main contributions of this paper are as follows:

- Examining simple network metrics can enable us to identify the source of performance failures, provided that the failure does not violate the flow constraints of the group communication protocol.
- Performance failures in a token-ring based group communication protocol yield few system-wide indicators of failure before the entire system begins showing degradation due to a single fault.

2 Related Work

Research into fingerpointing currently focuses on correlating system metrics or identifying faulty application level components on a causal path. Pertet *et al.* [8] use system and group communication level metrics such as CPU load and packets per second to identify the root cause of faults in the system. After a failure has been identified; metrics from multiple nodes are compared to see if any one node behaves differently from the rest of the nodes in the system. Machine-learning and time-series techniques are used to perform fingerpointing. We leverage the same data set in this paper.

Other research focuses on identifying faults on the causal request path. Aguilera *et al.* [1] identify performance problems by treating components as black-boxes by obtaining message-level traces of system activity.

Chen *et al.* [5][4][3] use runtime requests on the causal path between components. Pinpoint [5] uses data-mining techniques to correlate suspected failures and successful completion requests to determine which components are likely to be at fault. Chen *et al.* [3] perform statistical analysis on the collection of path traces to identify significant deviations from the normal behavior. Kiciman and Fox [6] further extend the Pinpoint system by building reference models of component interactions based on historical behavior.

Many of these approaches examine request or message traces on the causal path. However, we have previously observed that fault manifestations do not always restrict themselves to the causal path [8]. By only examining the causal path, we may miss fault manifestations outside that path; considering nodes outside the causal path may allow us to more quickly pinpoint the faulty node in the system.

3. Architecture-Driven Fingerpointing

3.1 Background

Spread group communication protocol uses a token-ring approach in order to guarantee reliable, ordered message delivery. Token-ring protocols impose a logical ring on the set of nodes in the group. A unique message, called a token, circulates within the group in a fixed path using point-to-point UDP messages. A node is only allowed to broadcast messages, using UDP broadcast, if it holds the token. After each node broadcasts its messages, it passes the token to the next node in the ring. A simplified example of a five node group, using Spread, is shown in Figure 1.

3.2 Architecture-Based Fingerpointing

The token-ring group communication protocol that we studied did not lend itself well to black-box path-based fingerpointing, as all nodes would experience a performance fault simultaneously, see Figure 2. However we also investigated if exploiting the constrained architecture of the system might help to fingerpoint the performance faults. If we could use the architecture, one of the first questions to address is whether we can pick a general metric that will allow us to fingerpoint virtually all performance failures in the system. Many metrics simply do not make sense for this type of analysis. For example, monitoring available memory or context switches on each node will not allow us to fingerpoint hardware network failures. At the same time, monitoring only packets per second will not allow to fingerpoint failures caused by faults that affect system resources like available memory, as the faults cause a system wide drop in network traffic

Other metrics however, do allow us to look more generally at where the system is experiencing failures that affect a node's ability to pass traffic. In particular, if we focus on counting sent and received tokens per node (basically a

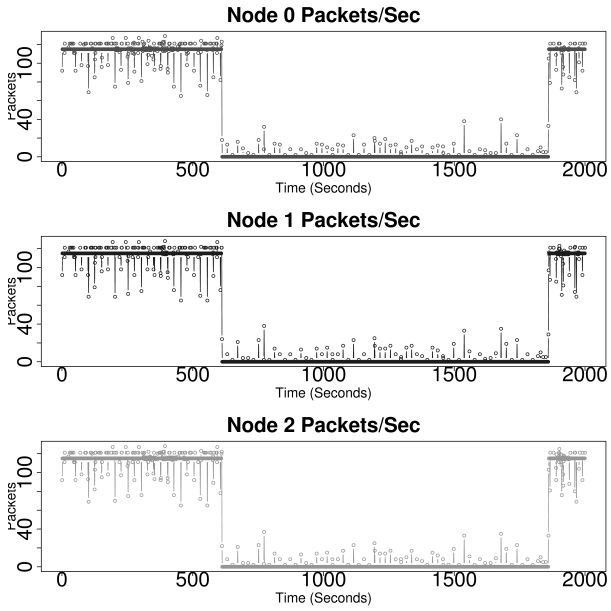

Figure 2. Network traffic at Nodes 0-2 during a 20% packet-loss fault. All nodes experience the fault at the same time

token-traversal count), combined with our knowledge of the architecturally-constrained network, we can obtain a much clearer picture of where the network is experiencing problems. We expect that as the leader node sends more tokens the nodes between the leader node and the faulty node will show an increase in token-traversal count, while nodes downstream from the faulty node will show little increase in the token-traversal count.

4. Experimental Setup and Data Collection

The data set used in this experiment was collected on a replicated, distributed testbed by Pertet *et al.* [8]. A full description of the experimental testbed, methods, and fault injection framework can be found in [8].

To summarize, the experimental test bed consisted of five nodes (four servers and one client). The servers were state-machine replicated over Spread using the MEAD fault-tolerant middleware that supports the state-machine replication of CORBA servers [7]. The experiment was conducted in the Emulab distributed test bed [9]. Each node consisted of an 850 MHz processor, 256KB cache, 512 MB ram, RedHat Linux kernel 2.4.18, and was connected by a 100 Mbps LAN. The testbed consisted of one client and four servers, each on its own node. The client sent 1024-byte requests every 10ms. Each run consisted of 30,00 round-trip client requests and ran for approximately 10 minutes.

The metrics were monitored under fault-free and faulty conditions. The overhead for the metric collection was reported to be a 28% increase in response times for the Spread traffic. The metrics collected on each node of the test bed for each run were:

- CPU usage(%): Percentage of time the CPU is busy executing user and kernel tasks, calculated every second
- Available Memory (bytes): Sum of the node's free and cached memory, recorded every second
- Context Switch rate (per second): Number of context switches occurring on the node per second.
- Packets (per second): Number of packets seen by the *libpcap* library on that node

Faults were injected at runtime into the system using the linker's library inter-positioning capability [8]. This allows faults to be injected transparently into the system. All faults were injected into node 4, which ran one of the replicated servers. The following faults were injected into the system:

- Memory leak: A memory leak was injected by bypassing the replica's *free()* system call using the interceptor. The server was modified to allocate 96KB with each request.
- Process hang: The replica's *read()* call was intercepted and delayed for several minutes.
- Packet-loss: The *send()* and *recv()* calls were randomly intercepted and messages were dropped. This dropped incoming and outgoing packets. Each call had messages dropped at a rate of both 1% and 20%.
- Crash Fault: A replicated node was abruptly shut down during system execution.

Each fault was injected five times into the system during various runs, along with five fault-free runs, giving a total of twenty-five system traces, of which twenty were faulty.

5. Results

Each faulty trace was analyzed by our simple fault detector, which monitored the metric of packets per second in order to give us an approximate time of when the fault began to disrupt the system. The network traffic was then analyzed by looking at the number of received and transmitted tokens per node during a three second window starting at the time that the anomaly was first detected. The crash and process hang faults did not trigger the anomaly detector as the group communication protocol was able to handle the fault with virtually no change in the network traffic. However, all of the memory leaks and packet-loss faults significantly impacted the network operation of the system.

5.1 20% and 1% packet loss

The faulty node was identified in all of the ten system traces where the node was randomly dropping 20% and 1% of packets that it recieved. In all cases, the faulty node and its preceding node had the lowest aggregate token-traversal count, while the other nodes had approximately the same token-traversal count. Given that the token traverses in a known direction, examining the sent token counts always identified the node as the source of the failure. The maximum difference between the healthy and the faulty nodes in tokens over a three second period was eight, while the minimun was six.

5.2 96KB Memory leak

The faulty node was also identified in all of the five system traces where the node did not free up all of the allocated memory. In all five cases, the faulty node had the lowest aggregate token-traversal count. The maximum difference between healthy nodes and faulty nodes in token count over a three second period was 76, while the minimum was 20.

5.3 Other Observations

Another interesting observation was that the faulty node could also be fingerpointed by decomposing the aggregate token-traversal count into received and sent tokens. A link on both sides of a node was suspect if the node's aggregate token count was significantly different from the other token counts (even if it was not the lowest). Taking this approach caused several of the links to be identified as potential problems in the system. However, using the fact that the tokens traverse in a known direction, links were analyzed by examining the sent and received tokens on a given link. If the sent count from the sending node and the received count from the receiving node were both normal, then that link was eliminated as a suspect link. If this approach is taken, then the links on either side of the faulty node are always left as the only suspect links, isolating the faulty node.

6. Conclusion

The communication infrastructure that enable key services in a distributed system are also often the source of faults propagating through the system. We have shown that by exploiting the architectural network-flow constraints imposed on a replicated system that uses a token-ring based group communication protocol, we can correctly identify the source of performance failures in the system with accuracy. These faults are particularly hard to trace as the standard application path-based approach to tracking fault propagation does not work in these cases. We have also shown that by only monitoring network statistics, we are able to pinpoint the source of different faults, which frees us from the problem of trying to select the right metrics to monitor in order to catch various types of performance problems.

We intend plan to investigate other common group communication protocols to see if we can extend the particular case of the token-ring system, and apply our approach to other types of communication protocols. In particular, it is unclear if our approach can be applied generally or is restricted to the token-ring model.

References

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *ACM Symposium on Operating Systems Principles*, pages 74–89, October 2003.

[2] 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*, pages 327–336, New York, NY, June 2000.

[3] M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based failure and evolution management. In *Symposium on Networked Systems Design and Implementation*, San Francisco, CA, March 2004.

[4] M. Y. Chen, E. Kiciman, A. Accardi, A. Fox, and E. Brewer. Using runtime paths for macroanalysis. In *9th Workshop on Hot Topics in Operating Systems*, Kauai, HI, 2003.

[5] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *International Conference on Dependable Systems and Networks*, June 2002.

[6] E. Kiciman and A. Fox. Detecting application-level failures in component-based internet services. In *IEEE Transactions on Neural Networks: Special Issue on Adaptive Learning Systems in Communication Networks*, pages 16(5):1027–1041, September 2005.

[7] P. Narasimhan, T. Dumitraş, S. Pertet, C. F. Reverte, J. Slember, and D. Srivastava. MEAD: Support for real-time fault-tolerant CORBA. *Concurrency and Computation: Practice and Experience*, Invited submission 2003.

[8] S. Pertet, R. Gandhi, and P. Narasimhan. Group communication: Helping or obscuring failure diagnosis. Technical report, Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-06-107, 2006.

[9] 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 *ACM Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002.