WORLDS '05: Second Workshop on Real, Large Distributed Systems

San Francisco, CA
December 13, 2005

INFRASTRUCTURE

Summarized by Rik Farrow

■ ***Experience with Some Principles for Building an Internet-Scale Reliable System***

Mike Afergan, Akamai and MIT; Joel Wein, Akamai and Polytechnic University; Amy LaMeyer, Akamai

Joel Wein described Akamai's Content Distribution Network (CDN) as having 15,000 servers in 1,100 third-party networks, with a NOCC managed by a day crew of eight and a night crew of three. The focus of this paper is not on CDN but on Akamai's experience in its seven-year experiment: in particular, keeping its distributed system running using Recovery Oriented Computing. In a single day, it is not unusual to lose servers, racks of servers, and even several data centers. The base assumption is that there will be a significant and constantly changing number of component or other failures occurring at all times in the network. The development philosophy is that their software must continue to work seamlessly despite numerous failures.

Wein outlined six design principles, organized in two sets of three. The first three principles are to ensure significant redundancy, use software logic instead of dedicated pipes for message reliability, and use distributed control coordination. Wein then gave examples of how these principles aid in operation during failures. The next three principles have to do with software design: fail cleanly and restart, zoning (their term for their brand of phased rollout), and notice and quarantine faults. No software is perfect, and these principles have helped to catch faults in software or configurations. Sometimes faults do not show up until a change has been rolled out to many systems. While most aborted rollouts occurred during phase one (36), the next most commonly aborted rollout occurred at the world level (23).

During the Q&A, Armando Fox asked why, if Akamai stages rollouts, there were ever any world aborts. Wein answered that sometimes that was when the problem showed up, and it could be caused by hardware, order of events, or corner cases. Fox followed up by asking if this was the only way to tickle the bug? Wein answered that stupid mistakes caused many of the world aborts, followed by needing to run on 50,000 servers before the problem shows up. Paul Lu asked how much of the system is homebrewed? Wein answered that a lot of this is custom code, but they are open to using other people's ideas and try not to be religious about these things. Jeff Mogul commented that most companies try to get down to one person per server, while the Akamai approach is different. Wein answered that their design notices a problem in an automated way, detects it right away, and removes it automatically. They have large brute force redundancy.

■ ***Deploying Virtual Machines as Sandboxes for the Grid***

Sriya Santhanam, Pradheep Elango, Andrea Arpaci-Dusseau, and Miron Livny, University of Wisconsin, Madison

Sriya Santhanam presented this research into the use of VMs in distributed computing. As most research Grid computing projects will run code that cannot be trusted, this code poses a security challenge. VMs provide security and isolation, environment independence, finer resource allocation, support for a wider variety of jobs, and a flexible, generic solution. They used Xen for their project, as Xen adds very little overhead when running applications on Linux. The target environment was Condor, software that watches for idle workstations so they can be used in Grid computing.

Santhanam described four different sandbox configurations, starting with the least restrictive and going to a very restricted environment. Even the least restrictive version has Condor alone installed within the VM, but arbitrary programs can be executed, and Condor itself is still exposed to network attacks. In the next version, VM gets launched on demand, and eager whole file caching is used, so no network access is required. In the next version, system calls get executed on the submitting machine rather than on the local system, and the final sandbox configuration includes lazy whole file caching and remote system calls on the submitting machine. Santhanam then presented graphs comparing the performance of the difference sandboxes.

Sean Rhea asked why sandbox 1 showed such low overhead compared to the other versions. Santhanam answered that only in this version is the VM already running. All other sandboxes include the time to start the VM in their overhead. Armando Fox asked, which sandbox would you choose for your friends? For people you trust to run code, sandbox 1 is easiest, whereas sandbox 4 adds additional components and complexity. Rhea asked if only one job is run at a time, and Santhanam answered yes, because the goal was limited and focused on defense. Rhea asked if the VM gets flushed after running each job. Santhanam answered that these are student workstations, running in labs, so the focus is on protecting these machines.

- ***MON: On-Demand Overlays for Distributed System Management***

  Jin Liang, Steven Y. Ko, Indranil Gupta, and Klara Nahrstedt, University of Illinois at Urbana-Champaign

Jin Liang described the problems that can occur when running applications on the PlanetLab Grid: monitoring and control require connections from the many remote systems each to a separate process. Monitoring the status of remote applications—noticing if they have crashed, if they need to be restarted, or if all applications need to be stopped and a new version uploaded—has been difficult with the existing tools. The goal also includes software distribution to all nodes.

MON is a management overlay network that uses an equivalent of a spanning tree to aggregate the results of all commands and to distribute commands to all the nodes. The overlay network is built on demand when needed, and is simple, lightweight, and suited to management, irregular/occasional usage, and short/medium-term command execution. When execution completes, the overlay goes away. Each remote host runs one daemon process that not only executes commands, but also participates in the construction of the tree. The construction of the tree must itself be lightweight and satisfy the requirements of both status query and software distribution. Liang described research into the best method of tree construction, a combination of random tree construction followed by local selection of neighbors. The paper provides more details of tree construction.

The Q&A focused more on what MON can and can't do than on tree construction. Someone asked, how can you be sure that a response from a node is calculated exactly once? Liang answered that this is not a problem, as each parent aggregates responses from children and sends just one response to its parent. Someone else asked, how can you find nodes that aren't working properly? Jiang said that MON is not designed for this purpose, but is focused on reliable, occasional monitoring.

CHOOSING WISELY

Summarized by Jin Liang

- ***Supporting Network Coordinates on PlanetLab***

  Peter Pietzuch, Jonathan Ledlie, and Margo Seltzer, Harvard University

Jonathan Ledlie first briefly reviewed what network coordinates are. Network coordinates such as Vivaldi try to approximate delay between two nodes using a geometric space. Thus, they are a powerful abstraction for distributed systems. However, the delay between nodes is not static. There could be gradual changes as well as unpredictable, unusually large deviations. The authors used a moving minimum filter to deal with this problem. Specifically, at any time, the next delay is predicted as the minimum of the previous four measurements. The second problem with network coordinates is that the changes in network coordinates might cause expensive application-level adjustments. For this, some update filter is used. Specifically, the centroid of the starting coordinate window is computed. The application is notified about the change only when the current centroid of the coordinate window is significantly different from the starting centroid.

Ledlie also showed a movie that illustrates how the coordinates would change, with and without the link (moving minimum) and update filters. Their evaluation results are based on the delay measurement on about 270 machines on PlanetLab.

One audience member commented that the filters are similar to network time protocol (NTP), including the update filter (whether a node is trustable in NTP). Ledlie said he will look at the differences. Another audience member asked if it is possible to report distribution as well as coordinates to the application, so that the application is aware of how much variance there is. Ledlie said this is currently not in the system but can be added. Someone else asked if delay is correlated with load, and Ledlie answered that there is a correlation.

- ***Fixing the Embarrassing Slowness of OpenDHT on PlanetLab***

  Sean Rhea, Byung-Gon Chun, John Kubiatowicz, and Scott Shenker, University of California, Berkeley

  Awarded Best Paper!

There is a lot of hype about DHTs (Distributed Hash Tables). However, many previous results were obtained in benign environments (i.e., in lab). The authors of this paper wanted to improve the performance of DHT "in the wild," and by considering 99th-percentile performance numbers. Real-world applications may not have dedicated machines, and the authors want to provide an OpenDHT service. There are two flavors of slowness in nodes. The first is unexpected, which is discovered only when a request is routed to the node. The second is consistent slowness, which can be avoided by maintaining a history. The authors provided two solutions to node slowness: (1) Delay-aware routing, in which the delay to the next hop and the distance in the key space between hops are considered when selecting the route.This is in contrast to traditional DHT, where routing is purely greedy in the key space. (2) Parallelism. Using iterative routing, the requester can keep multiple outgoing RPC requests. Thus, even if some slow nodes are encountered, other requests can quickly get results. The user can also send the initial request to two different gateways.

Their results, obtained from PlanetLab using concurrent execution methods (i.e., a particular approach is randomly selected for lookup each time) show that delay-aware routing is clearly best, reducing the 99th-percentile latency by 30% to 60% without increasing overhead. Other techniques can also reduce the delay, but will increase overhead.

- ***(Re)Design Considerations for Scalable Large-File Content Distribution***

Brian Biskeborn, Michael Golightly, KyoungSoo Park, and Vivek S. Pai, Princeton University

Well-designed systems may not work efficiently in a real environment. In redesigning the Coblitz file transfer service, the authors achieved a 300% faster download and a 5x load reduction on the origin server. Coblitz uses a content distribution network for file transfer. A smart agent will divide the request for one file into multiple requests for file chunks. The requests are sent to different CDN nodes that have the chunks cached. There are several techniques that are used to improve the downloading. For example, some nodes are consistently slow, and these are removed. Also, when a node is slow, instead of waiting for time-out and retry, the new design keeps several connections to compete with each other. Also, before a node requests the file from the origin server, it looks at other nodes to see if they are more suitable for making the request. Using these techniques, the new Coblitz system's performance is much improved.

One audience member asked if the set of slow nodes is stable, because they have found it (in terms of delay instead of bandwidth) unstable. KyoungSoo said they have done a lot of bandwidth measurement and the set is stable. Another questioner asked where the bottleneck is for downloading, and KyoungSoo answered, the bandwidth cap. Another audience member asked whether they have run comparisons with SHARK, and KyoungSoo answered, yes, and with BulletPrime.

FROM THE TRENCHES

Summarized by KyoungSoo Park

- ***Non-Transitive Connectivity and DHTs***

Michael J. Freedman, New York University; Karthik Lakshminarayanan, Sean Rhea, and Ion Stoica, University of California, Berkeley

Michael Freedman started by pointing out the difficulty of running DHT applications due to the non-transitive connectivity problem. Non-transitive connectivity is A not being able to communicate with B, while A and C and also C and B can communicate. Because the conceptual DHT design assumes full connectivity, people must often resort to their own hacks to get around this problem. Non-transitive connectivity occurs for various reasons, such as link failures, BGP routing updates, and ISP peering disputes, and 9% of PlanetLab nodes are reported to show such phenomena, according to Stribling.

For DHTs, Michael classified the problems as "invisible nodes," "inconsistent roots," "broken return paths," and "routing loops." Node B is said to be A's invisible node if A can communicate with B via C, but not directly with B. A simple fix for this would be to let A add (or remove) B only when A directly communicates (cannot communicate) with B. In order to get over the performance impact of invisible nodes, Michael proposes (1) timeout estimates via network coordinates, (2) parallel lookups, and (3) caching of unreachable nodes. Another problem is "inconsistent roots," possibly caused by network partition. Two distictive nodes, say R and R', which cannot communicate with each other, may each act as if it were the root. An expensive consensus algorithm is one way of solving this; another is to use link-state routing among the leaf set with FreePastry 1.4.1. "Broken return paths" means that a direct return path between the destination node and the entry node may not exist, while a forwarding path in the DHT lookup does exist. One solution is to route backward along the lines of the forward path, and the other is one-node source routing via a leaf node randomly chosen by the destination node.

Justin Cappos asked whether such non-connectivity is mostly unidirectional or bidirectional, and Michael responded that he did not measure it, but he thinks it is mostly asymmetrical. Indranil Gupta mentioned that the problem is being solved by RON, and asked if the problem is the fundamental limit of DHT. Michael and Sean Rhea responded that it is a problem of whether to store more states in the routing table. Rick McGeer added that Tapestry has a backup path, and Sean confirmed that.

■ *Why It Is Hard to Build a Long-Running Service on PlanetLab*

Justin Cappos and John Hartman, University of Arizona

Justin Cappos began by asking why we do not see many long-running services on PlanetLab, even though PlanetLab was mainly created to support them. He divided the types of services that researchers are interested in into three categories. The first category consists of highly novel services that are publishable but unstable and that usually end up perishing right after publication (e.g., Bullet and Shark). Another category includes services such as AppManager and Sirius, which have high stability but are not novel enough to be made into papers.The last category, comprised of services that combine the two features, includes Stork, Bellagio, and CoDeeN. Justin explained that the reason why we do not see many research services on PlanetLab is because there is not much incentive to provide long-running services, which would take non-trivial maintenance time that cannot be rewarded with publication, and he emphasized the need to give more credit to long-running services.

He described the process by which Stork was shaped into a reliable, long-running service and the lessons to be drawn from Stork's example: the need to have a reasonable fall-back scheme for unreliable services, to build on other research services, to be aware of corner cases, and to use other research systems and provide the feedback essential to improving their quality and usability.

Jeff Mogul commented that it is not the novelty of the idea but the novelty of the results that draws the attention of paper reviewers, and conferences like OSDI mainly focus on such results. Sean Rhea commented that in the past, good projects all started with a simple scheme but evolved into a novel system by fixing problems in the middle.

■ *Using PlanetLab for Network Research: Myths, Realities, and Best Practices*

Neil Spring, University of Maryland; Larry Peterson, Andy Bevier, and Vivek Pai, Princeton University

Years of operation of PlanetLab have created various myths that used to be true. Still, some research folks believe PlanetLab is too flaky or too overloaded for some experiments. Neil Spring talked about what is and what is not true about PlanetLab, based on his careful observation.

He started with what's true. First, the experimental results are not reproducible on PlanetLab, because it is designed to provide a real-world Internet environment rather than a controlled testbed. Even so, short experiments can be measured more carefully by avoiding what CoMon has determined to be heavily loaded nodes. For reproducible results, Emulab and Modelnet can be alternatives to PlanetLab. Also, PlanetLab is not representative of the Internet or peer-to-peer network nodes, because PlanetLab cannot cover the entire Internet and its nodes are not desktop machines as in P2P systems. However, more and more traffic on PlanetLab includes lots of commercial sites and is not PlanetLab-exclusive. Although PlanetLab does not use P2P nodes, its nodes can be used as managed core nodes in P2P systems, as in End System Multicast (ESM).

Neil also enumerated myths that are no longer true. First, PlanetLab is no longer overloaded. Measurement shows that 20 to 30% of available CPU cycles are available at any given time, even right before major conference deadlines. The current per-slice scheduling prevents any single slice from hogging all the CPU cycles. Another myth is about PlanetLab's supposed inability to guarantee resources, but resources are available because they are managed by a brokerage service, especially in running short-term experiments.

Best practices also help in demystifing some myths and in improving the reliability of the experiments on PlanetLab. By using kernel timestamps and instrumenting traceroute one can time the packets on PlanetLab with great accuracy, and measuring bandwidth via precise packet trains is still possible with the use of an appropriate system call such as `nanosleep()`. Random site measurement restriction imposed by PlanetLab AUP can be implicitly lifted by soliciting outside traffic, and observation of such rules is easily achieved by using a service like Scriptroute. Finally, it is beneficial to know that surviving excessive churns is essential in making long-running experiments.

Mic Bowman asked if PlanetLab suffers from memory pressure due to the large memory footprint of some Java programs. In response, Vivek mentioned that a recent memory pressure test shows that 80% of all PlanetLab nodes have at least 100MB available. Besides, pl_mom is effective in maintaining a good level of memory status by killing the highest memory consumer when memory pressure arises.

Sean Rhea commented that one problem is that people tend to try a random tool that works on a stock Linux, but get frustrated to see it not working on PlanetLab. He also mentioned that it would be useful to make the packet trains into a tool like KyoungSoo and Vivek's CoTop. Neil responded that using Scriptroute will provide accurate timing without carefully implementing it individually.