;login:

THE MAGAZINE OF USENIX & SAGE

June 2003 • volume 28 • number 3

inside:

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

4th USENIX Symposium on Internet Technologies and Systems (USITS '03)

Seattle, Washington

March 26-28, 2003

[Only a few reports were received on this conference – Ed.]

SESSION: ROBUSTNESS

Summarized by Ajay Gulati

Why Do Internet Services Fail, and What Can Be Done About It?

David Oppenheimer, Archana Ganapathi, and David A. Patterson, University of California, Berkeley

David Oppenheimer and his group studied various causes of failures for Internet services and the effectiveness of various techniques used to mask service failures, as part of their recovery-oriented computing project currently going on at University of California, Berkeley, and Stanford. Today, Internet services have 24/7 expectancy from users, and in spite of a lot of techniques used by the designers for higher availability, they still fail, although such failures are not always directly visible to users. He talked about the difficulties users face in convincing service providers to allow access to their problem-tracking databases, saying "Nobody wants their failure information to be public."

They studied three large-scale services, which he classified as "Online" (a mature service/Internet portal), "Content" (content-hosting service), and "ReadMostly" (mature readmostly Internet service). They got access to problem-tracking databases on the first two and a log of user-visible failures on the third. Oppenheimer made a clear distinction between component failures and service failures: a service failure is one which is visible to end users; component failures are sometimes masked by redundancy and do not cause service failure.

According to Oppenheimer, operator errors were found to be the leading cause of failure in two of the three services. Most of these errors were due to misconfiguration and post-installation changes made by the operators. He showed that operator errors are the largest contributors both in terms of numbers and time to repair (TTR), contributing approximately 75% of all TTR on both online and content services. Oppenheimer also observed that the highest proportion of operator error eventually became visible to users as compared to any other type. In readmostly services, network errors dominated operator errors and caused 76% of all the service failures. Oppenheimer attributed that to simple and more robust application software and less need for day-to-day maintenance on the part of operators.

Oppenheimer went on to explain various techniques commonly used to mitigate failures, such as online correctness testing, exposing/monitoring failures, redundancy, config checking, and online fault injection. He presented three techniques, namely, component isolation, proactive restart, and pre-deployment correctness testing, which are not currently in use but which could have prevented some failures from occurring. He stressed that most of the techniques work well to mask hardware, software, and network failures, but that we lack efficient techniques to mask/detect operator failures. Also, operator errors are difficult to diagnose and detect before they convert into failures.

Finally, Oppenheimer stated some of the difficulties in extracting data from problem-tracking databases, since data entered into them is sometimes incorrect and cannot be analyzed by writing just a few database queries. David said that a global repository of common errors, and what was done to handle them, might be of great help in such research.

Using Fault Injection and Modeling to Evaluate Performability of Cluster-Based Services

Kiran Nagaraja, Xiaoyan Li, Ricardo Bianchini, Richard P. Martin, and Thu D. Nguyen, Rutgers University

Thu Nguyen started off by saying that today unavailability costs are really high and even 99.9% of availability is not sufficient for Internet services in some cases. Most of the large Internet services use large clusters of commodity computers as their infrastructure. These services are often quite complex and have large design space. Measurement of availability is mostly based on a practitioner's experience and intuition rather than a quantitative methodology. Nguyen proposed a metric combining performance and availability. Before going to the two phases of the metric, he explained a seven-stage piecewise linear model showing various stages that a server goes through, from fault occurrence to recovery: (1) component fault occurs, (2) fault detected, (3) server stabilizes (with performance degradation), (4) component recovers, (5) server stabilizes (still not performing at peak), (6) operator resets, (7) normal operation.

In the first phase, they tried to measure the system's response to individual faults. For the second phase, evaluators need to use an analytical model to combine expected fault load for the server with the measurements taken in the first phase. This gives them a sort of dot product of faults and behavior vectors. To demonstrate the effectiveness of the methodology, the authors studied performability of four versions of PRESS (a highly optimized yet portable cluster-based locality-conscious Web server) against five classes of faults associated with network, disk, node, and application. In phase two of the case study, they compared performability of various versions of PRESS and showed how the model can be used to evaluate various design tradeoffs, such as adding RAID or

increasing operator support. Finally, Nguyen discussed some of the lessons learned after applying their methodology to PRESS.

Mayday: Distributed Filtering for Internet Services

David G. Anderson, MIT

Denial-of-service attacks are quite common these days; it doesn't require a high degree of sophistication to implement them. Many measures have been suggested to prevent and avoid DoS attacks, but for various reasons none of them has been globally deployed. Either they require lots of changes in routers, affect normal operation of the server, don't provide any guarantees about immediate relief to the deployer, or take too much time to recover once the attack is detected. Anderson proposed Mayday, an architecture that provides pro-active protection against DDoS attacks, imposing overhead on all transactions to actively prevent attacks from reaching the server.

He made it clear in the beginning that Mayday works only for flooding attacks and not for other smart attacks that could potentially crash a server with incorrect data. He also pointed out that an attacker who can watch all traffic in the network is too powerful to resist; one who targets a particular node or can watch only a part of network traffic, however, can be eliminated, basically because the server has time to detect and prevent the attack before the attacker gets hold of all the nodes. Mayday combines overlay networks with lightweight packet filtering that is efficiently deployed in routers around the server. This filter ring of routers actually provides Internet connectivity and a set of overlay nodes that can talk to the server via the filter ring. Clients communicate with overlay nodes using some application-defined client authenticator. Overlay nodes authenticate the client, perform protocol verification, and then send it to the server through the filter ring using a lightweight authenticator. According to Anderson, this leaves the designer with a lot of choices to trade off among security, performance, and ease of deployment. The performance and robustness of the resulting system depend heavily on overlay routing techniques and the authentication mechanism.

Anderson went on to discuss the pros and cons of various lightweight authentication techniques such as the use of the server's destination port/address, overlay node source address, or any other field in the header for authentication. Similarly, in the case of overlay routing, one can give access privileges to all overlay nodes or only to some of them. In the latter case, access can be by: (1) indirect routing – overlay nodes pass the message to a particular node which has access to the server; (2) random routing – the message is propagated randomly in the overlay until it reaches a node having access to the server; (3) mix routing – each node knows the next hop but not the final destination. Fake traffic can be generated between overlay nodes to confuse the eavesdropper. Once such a protection mechanism is in place, the server can switch between normal mode of operation and secure mode when an attack is detected.

The paper then covered some of the practical attacks that can be used against such filtering-based schemes. Probing or timing attacks, for example, can quickly determine a valid lightweight authenticator and use that to pass through the filter ring. Finally, some of the more sophisticated attacks that might get control over an overlay node and launch internal attacks in the overlay were discussed.

Q: What can Mayday do about attacks that are not well known?

A: It's always possible to come up with some attacks that are not taken care of. But my work mostly concentrated on flooding and DDoS attacks, and it is quite sufficient to handle them.

Q: How long does it take to change the configuration and other things in a router so as to avoid attacks?

A: It depends on how much of a window you want to allow for attackers to detect and cut through the security system. But automated tools these days take minutes, and sysadmins can do this in a matter of hours.

SESSION: RESOURCE MANAGEMENT AND SCHEDULING

Summarized by Ajay Gulati

Adaptive Overload Control for Busy Internet Servers

Matt Welsh and David Culler, University of California, Berkeley, and Intel Research

Matt Welsh started off by saying that 9/11 has reminded us of the inability of most Internet services to scale and handle spikes in demand dynamically. Peak workload may be orders of magnitude higher than the average, and managing the performance of a server under such conditions becomes really difficult. Most of the common approaches apply strict limits on resources, such as bounding the number of open sockets or threads or limiting the maximum CPU utilization. He stressed the point that these limits should in some sense be representative of user response time and not just the characteristics of the servers.

Overload management techniques are based on SEDA (staged event-driven architecture), which is a model for scalable and robust Internet services. SEDA decomposes a service into a graph of stages, where each stage is an event-driven service component. Each stage uses a small, dynamically sized thread pool to handle some aspect of request processing. These stages are connected via explicit queues that act as a mechanism for control flow between the stages and a boundary between them. Each stage's incoming event queue is guarded

by an admission controller that accepts or rejects new requests for the stage. Each stage can do dynamic resource control, to keep itself in its normal operating mode by tuning parameters for its operation, such as changing the number of threads based on the workload and performance of the stage. Welsh showed a code snippet to demonstrate that overload management is built into the application itself, as any stage can reject a queue request if it feels that accepting it might lead to performance loss. He discussed some of the alternatives to rejecting requests for load shedding. One might start working at degraded performance and tell the user, "This will take time, please wait," as most airlines do. Another way would be to send explicit rejections, such as "We are busy now, try again later." Some Web sites also do some social engineering by sending error messages, saying, for example, "Zipcode is wrong," just to confuse users and gain more time to handle the request. This scheme allows overload control to be performed in response to measured bottlenecks, which is better than having an external control based on general service capacity. Also, handling rejected requests can be done on a stage basis, since the application knows which stage was bottlenecked for a given request. Welsh presented three mechanisms of overload control in SEDA:

1. Performance metric: A 90th percentile response time is used, which is a realistic and intuitive measurement of client-perceived system performance. The response time value may be set by the system administrator and might depend on request type – for example, not kicking out a user with lots of stuff in a shopping cart.

2. Response-time controller design: The controller associated with each stage observes a history of response times and throughput of the stage, and adjusts the rate of acceptance for new requests to meet the goals of performance.

3. Class-based differentiation: This scheme can prioritize requests from certain users over others and handle service level agreements based on what different clients are paying for the service. The authors developed a Web-based email service, which was a clone of Yahoo mail, allowing users to access and manage their emails. During large load spikes, SEDA compared favorably to servers with fixed connection limits in meeting 90th percentile targets. Also, SEDA's rejection rates were lower than other approaches. Finally, multi-class service differentiation led to different rejection rates for various classes, with requests from one class meeting the 90th percentile response time more frequently than the other, less-preferred class.

Q: Is this architecture designed for a single machine or a cluster?

A: It will work very well on a cluster as well. In that case, different stages might be implemented on different machines.

Model-Based Resource Provisioning in a Web Server Utility

Ronald P. Doyle, IBM; Jeffrey S. Chase, Omer M. Asad, Wei Jin, and Amin M. Vahdat, Duke University

Summarized by Ajay Gulati

Jeff Chase started off by saying that provisioning of shared resources at large-scale network services is one of the biggest challenges for system research today. The authors focused on automation of on-demand resource provisioning for multiple services hosted by a shared server infrastructure competing for resources such as memory, CPU time, and throughput from storage units. A slice of the these resources is allocated to each service to meet service quality targets decided in SLAs. Chase presented a novel model-based approach, in which internal models of service behavior are used to predict initial resource allotment and are changed dynamically using a monitoring and feedback system. He introduced the notion of a "utility operating system" that handles resource management across the utility as a whole. The authors observed that network service loads have been studied extensively, have common properties, and can be represented by models fairly accurately. Chase went on to discuss some of the models derived from basic queuing theory and showed that behaviors predicted from these models were quite similar to actual observed behaviors. Resources such as memory, storage I/O rate, and response time were closely modeled by parameters like requests/s, average object size, average CPU demand/req., memory size for object cache, and peak storage throughput in IOPS.

Once the models were obtained, a resource provisioning algorithm was used to plan least-cost resource slices and get an "allotment vector" for each service, representing CPU, memory, and storage allotments. This algorithm consists of three main primitives:

1. Candidate – plans initial allotment vectors that are guaranteed to meet SLA response time targets for each service. It does not consider resource constraints, which is done by the other two primitives.

2. LocalAdjust – takes a candidate allotment vector and request arrival rate as input and outputs an adjusted vector adapted to local resource constraint or surplus exposed during initial assignment. It basically constructs an alternative vector that meets target within the resource constraints.

3. GroupAdjust – works on a set of candidate vectors to adapt to a resource or a surplus exposed during assignment to meet system-wide goals. For example, it can reprovision available memory to maximize the hit ratio across a group of hosted services.

Chase presented various graphs showing how these primitives allocated surplus memory to optimize global response time and flexibility of the model-based

approach in adapting to changes in load or system behavior. Finally, he presented the evaluation methodology for the technique, for which they used a cluster of load-generating clients, a reconfigurable switch, DASH Web server, and network storage servers accessed using DAFS (direct access file system). The results showed that the predicted and observed resource utilizations were fairly close and that model-based provisioning is quite effective for resource management on cluster utilities.

Conflict-Aware Scheduling for Dynamic Content Applications

Cristiana Amza, Alan L. Cox, Rice University; Willy Zwaenepoel, EPFL

This work focused on scaling a dynamic content site (e.g., Amazon.com) through a new technique called conflict-aware scheduling. Dynamic content sites consist of three tiers: Web server, application server, and database. The need for scaling the main site arises because there may be many clients accessing such sites. Replicating the front tiers is easy because they do not contain the dynamic content; the data that changes is stored in the database. Currently, state-of-the-art dynamic-content Web servers rely on a single very expensive database supercomputer to satisfy the volume of requests. The solution introduced was to scale the database tier by using replication on clusters. This allows a low-cost solution and, most importantly, incremental scaling and strong consistency at the same time. Amza justified the choice of replication based on characteristics of dynamic-content applications such as locality (hot-spots in the workload) and higher read-query complexity as compared to the write-query complexity. On the other hand, traditional replication has a known down side. It is well known that one cannot get both scaling and strong data scheme (for consistency), asynchronous writes (for scaling), and conflict avoidance (for improved scaling).

The TPC-W e-commerce benchmark with its three workload mixes – browsing, shopping, and ordering – was used to evaluate conflict-aware scheduling. The group used commodity hardware and software components in the evaluation: the Apache Web server and PHP module for the Web and app servers and the MySQL database engine. The only correct protocol that could previously be used to satisfy TPC-W's requirement for strong consistency was the eager protocol with synchronous writes. The scaling of the eager protocol is poor and gets worse with increasing writes in the mix and larger clusters. Amza's results showed that conflict-awareness compared favorably to both eager and conflict-oblivious lazy replication over a large range of cluster sizes and conflict rates. Scaling was close to ideal in the conflict-aware protocol for the browsing and shopping mixes of TPC-W up to large cluster sizes. The ordering mix was a pathological case for the protocol. It had a very high fraction of writes (50%) and consisted mostly of ordering transactions, which were very long and held locks on all useful tables. The scaling for this mix flattened at 16 databases, although conflict awareness still brought significant improvement over eager, which did not scale at all in this mix.

INVITED TALK

Fast, Reliable Data Transport

Michael Luby, Digital Fountain, Inc.

Summarized by Xuxian Jiang

An interesting talk on the data transport issue.

Luby first examined the weakness existing in traditional data transport (UDP, TCP etc), especially the interconnection between rate control and reliability mechanism in TCP data transport. In traditional data transport, the rate control is highly constrained by the quantity of unacknowledged data allowed, and loss estimation is mainly based on acknowledgments received.

Secondly, he talked about the digital fountain data transport approach, which decouples the relationship between reliability and flow/congestion control. The reliability is provided without using feedback. Such feedbackless-based reliable data transport has many desirable features in data transport: (1) speed over large distances and loss networks; (2) predictable speedy control of data transport; (3) global transport with local performance; (4) massive scalability; (5) flexibility in choosing a flow/congestion control mechanism.

Such an approach can be widely adopted in many situations: (1) wireless and satellite communication; (2) enable receiving when receivers have intermittent connectivity; (3) enable data transport even in highly unreliable communication, which may experience unknown or variable loss.

The digital fountain approach is analogous to a water fountain: it doesn't matter what is received or lost, it only matters that enough is received. The sender sends encoded data at the rate decided by the selected flow control mechanism and the receiver receives some of the encoded data and is able to reconstruct the original data. It is not necessary for the receiver to provide feedback for the transport, what really matters is that enough information is received. The encoding and decoding technology does the essential core of the magic.

There are serveral erasure codes which can succeed in the encoding for this purpose, including Reed-Solomon codes (1960, Reed and Solomon), Tornado codes (1997, Luby et al.) , LT codes (1998, Luby), and Raptor codes (2001, Shokrollahi). The common properties of these digital fountain codes include: (1) encoding only as long as data flows; (2) recoverability of data from required encoding, (3) low complexity for encoding and decoding; (4) ability to encode

very large data; (5) ability to produce an unlimited flow of encoding.

The pros and cons of example digital fountain codes were examined in terms of data length, encoding length, flexibility to receive from multiple sources, memory requirement, computational work, reception overhead, failure probability, etc. Interested readers are referred to the corresponding encoding papers, especially LT codes (1998, Luby) and Rapter codes (2001, Shokrollahi).

The talk concluded with some typical and interesting application scenarios and on-going projects, such as CINC deployment, broadcasting data to automobiles, and robust communication in challenged environments.