conference reports

First Symposium on Networked Systems Design and Implementation (NSDI '04)

San Francisco, California March 29-31, 2004

TECHNICAL SESSIONS

SENSOR SYSTEMS SESSION

Summarized by Ramakrishna Kotla, Xun Luo, and Vinay Mallikarjun

The Emergence of Networking Abstractions and Techniques in TinyOS

Philip Levis, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler, University of California, Berkeley; Sam Madden, MIT Computer Science and Artificial Intelligence Laboratory, and Intel Research Berkeley; David Gay, Intel Research Berkeley

Is sensor system design any different from conventional mobile system design? Sam Madden explained that these two kinds of systems differ in that sensor systems are limited by resources like memory (512–4K bytes), battery power, etc. "Bluetooth is totally wrong for sensor systems, as power management is totally integrated into the protocol and does not expose it to the application." He explained the design of the TinyOS operating system, used in sensor systems that they built at Intel Labs, and how it was different from traditional OSes. TinyOS uses component-based design with no "kernel" and a single application running at a time, which chooses its own set of OS services. He then explained the software abstractions of the resources (energy, memory, etc.), the various applications built on top of TinyOS (localization, habitat monitoring, etc.), the services provided in TinyOS (network stack, radio stack, mica stack), and the issues involved in building these services. He concluded the talk by making the observation that software abstractions in sensor systems are dictated by limited memory constraints, energy concerns, flexibility for application to choose policies, and the noninteractive nature of applications, which makes it different from traditional mobile systems.

Q: The first generation of sensor systems are not powerful, but with powerful future sensor systems will the abstractions change?
A: Some of the constraints and issues with sensor systems are fundamental, like energy concerns, which will hold in future, and these abstractions hold good for them as well.

Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks

Philip Levis and David Culler, University of California, Berkeley, and Intel Research Berkeley; Neil Patel, University of California, Berkeley; Scott Shenker, University of California, Berkeley and ICSI

This won the Best Paper award at the conference. Re-tasking nodes in the sensor systems requires efficient dissemination of data. Maintenance of the system could be costly since it calls for transmitting 20–400 bytes of data for each update. Philip Levis presented a gossip-based algorithm called Trickle for propagating and maintaining code in sensor systems. The algorithm assumes the presence of a broadcast medium and that nodes can verify metadata. This

Philip Levis

This issue reports on the First Symposium on Networked Systems Design and Implementation (NSDI '04) and on the 3rd USENIX Conference on File and Storage Technologies (FAST '04).

For NSDI '04: Our thanks to Amin Vadhat, who shepherded the following summarizers:

Magdalena Balazinska
Laura Grit
Vinay M. Igure
Suchita Kaundin
Chip Killian
Ramakrishna Kotla
Xun Luo
Vinay Mallikarjun
Piyush Shivam
Chunqiang Tang
Praveen Yalagandula
Aydan Yumerefendi

For FAST '04: Thanks to Ismail Ari and his summarizers:

Michael Abd-el-Malek
Nitin Agrawal
Akshat Aranya
Dean Hildebrand
Andrew Klosterman
Xun Luo
Kiran-Kumar Muniswamy-Reddy
Steve Schlosser
Shafeeq Sinnamohideen
Deepa Tuteja
Wenguang Wang
Lan Xue
Aydan Yumerefendi

Note: The reports on BSDCon '03, held in San Mateo, California, September 8–12, 2003, can be found at *http://www.usenix.org/events/bsdcon03/confrpts.pdf*

algorithm uses a "polite gossip" policy in which the nodes periodically broadcast a code summary to neighbors but stay quiet if they hear a summary identical to theirs. It uses a sender-side rate-controlling mechanism to avoid flooding the network: nodes hear a small trickle of packets that can keep them up-to-date. Philip went on to explain how they evaluated the system using a TOSSIM simulator and the real system. He concluded the talk by presenting results, which show that Trickle scales logarithmically with respect to the density of the nodes and can achieve rapid propagation with low maintenance.

Q: How does it scale with a large amount of data propagation?
A: For large data, a hierarchy of metadata suppression is used to scale, where the only differences in data are sent.
Q: Lossy links create heavy tail distribution in the results. Can we increase the load probabilistically so that there are no bottlenecked links?
A: We can identify critical links and retransmit more often on those links.

Programming Sensor Networks Using Abstract Regions

Matt Welsh and Geoff Mainland, Harvard University

Programming sensor networks is difficult in that data needs to be pushed into the network, as opposed to pulling out the data and processing it at a centralized node. Matt addressed the question, "How do we develop a programming model for sensor networks as a whole?" He raised issues that the programming model has to take care of, like allowing accuracy/overhead tradeoff, flexible communication primitives, exposing certain resource parameters to the application while hiding others, etc. He defined "abstract regions" as a group of nodes with some geographic or topological relationship that capture common idioms in sensor networks. Various operations can be performed on abstract regions, like neighbor discovery, accessing shared variables, and reductions to support aggregation of shared variables. He then explained the applications he built using these primitives: contour finding, directed diffusion, and object tracking. Region operations are inherently statistical, and programming abstractions allow tuning these variables in order to trade resource usage for accuracy. He concluded the talk by making a case for a spatial programming language using abstract regions to program sensor networks that are inherently resource constrained, volatile, and distributed. For further details on this work, please visit *http://www.eecs.harvard.edu/~mdw/proj/mp*.

Q: Why can't we use a distributed programming paradigm for sensor networks?
A: Nodes in sensor networks are resource constrained, which requires a different programming methodology.
Q: Ensemble programming is a good thing irrespective of resource constraints.
A: I agree.
Q: Why is exposing certain parameters necessary in sensor systems, unlike traditional systems?
A: Programmers think at a high level; however, they need knobs to adapt to application requirements like the tradeoff between accuracy and lifetime.

NETWORKING SESSION

Summarized by Laura Grit and Xun Luo

Design, Implementation, and Evaluation of Duplicate Transfer Detection in HTTP

Jeffrey C. Mogul, Terence Kelly, HP Labs; Yee Man Chan, Stanford Human Genome Center

Terence Kelly presented this paper addressing the problem of duplicate transfers in HTTP. Redundancy can be foiled due to aliasing, rotation, and faulty metadata. To demonstrate that this problem exists, the authors ran two large traces on Compaq and WebTV and found that one in five transfers is redundant. The authors' solution is to create a digest of payloads (the entire message body of the HTTP response) and to cache the payload in a digest in a method they call Duplicate Transfer Detection (DTD). Their algorithm involves the client asking the digest for a requested URL. If a match is in the cache, the server delivers the cached payload; if it is not in the cache, the client needs the full response from the URL. They argue that DTD eliminates all redundant transfers.

Their talk focused on the analytic and benchmark results of their work. The authors found their method reduces latency when response length is sufficiently long. The overheads of their implementation were under 2%. When asked if they used different benchmarks, Terence referred the audience to the paper to see results with different Web service payload models. With regard to benchmarking, their response time and time to first byte were best with small payloads, like that of cell phones. The cost of a DTD miss is constant except for large body sizes, and they pay an extra round trip for misses. However, the benefits of DTD increase with body size.

Code is available for their DTD implementation over Squid at *http://devel.squid-cache.org/dtd*, but they warn that the code is currently too buggy for production use.

OSPF Monitoring: Architecture, Design, and Deployment Experience

Aman Shaikh and Albert Greenberg, AT&T Labs–Research

Aman Shaikh presented their experiences with OSPF (Open Shortest Path First) monitoring. Their objective was to perform both real-time and offline analysis of OSPF behavior. They created a monitor that collects OSPF Link State

Advertisements (LSAs) passively from the network. There are three ways their monitor can attach itself to the network: multicast group, full adjacency mode, or partial adjacency mode. In particular, they are looking for topology changes, node flops, LSA storms, and anomalous behavior. They have deployed their monitor in both an ISP and an enterprise network.

In implementation, the monitor consists of three components: LSA Reflector, to capture LSAs from the network; LSA aGregator, to perform real-time analysis; and OSPF Scan, for offline analysis. The monitor infers what the network looks like from the messages being sent. If it detects anomalies, it sends alarms to a higher-level network manager. By doing this, they can detect things that other management systems cannot find. For example, they have caught equipment problems, configuration problems, and even an OSPF implementation bug. Since they keep all information, their offline performance evaluator can make their offline simulator think there is an actual large topology and can accurately determine what is happening on the network.

When asked about the difference between partial and full adjacency modes, they said that, in their full mode, LSA Reflector is more unstable. Aman was also asked to explain what OSPF monitoring does not do. He said that OSPF monitoring does not do everything, but it can tell people if the problem is in OSPF and whether it may need other monitoring when OSPF is not running.

OverQoS: An Overlay-Based Architecture for Enhancing Internet QoS

Lakshminarayanan Subramanian, Ion Stoica, and Randy Katz, University of California, Berkeley; Hari Balakrishnan, MIT

Lakshminarayanan argued that the current best effort is not sufficient for QoS applications. Deployment of techniques is difficult because the IP layer is hard to change – you need everyone to agree on an implementation – and requires end-to-end deployment. They believe that overlay networks can do the same thing for QoS as they did for Multicast.

In this overlay environment, there is no control on the cross traffic or on node placement. Their implementation must be fair to cross traffic and keep network stability when there are multiple QoS overlays. Their technique is to trade resources between overlay flows to manipulate QoS parameters. For example, they would sacrifice throughput for better loss characteristics. Their design ensures that network overlay traffic looks like TCP traffic and follows TCP guarantees. To do this they add controlled-loss virtual links to characterize the service for the overlay and to bind observed loss rate. They keep normal flows the same across the link, but use the rest of the space on the link for service. These overlays can be used with multi-player games, streaming media, or leasing overlay networks. In their experimentation, they found OverQoS can provide statistical loss and bandwidth guarantees, it is fair and stable to coexist with TCP, and cost overheads are only between 0.5 and 6%. With 99% probability, available bandwidths were at least the QoS value.

When asked what is the additional overhead of OverQoS, they said it is the bandwidth added to the stream to reduce loss rates. The follow-up question was, What if everyone starts sending multiple packets? In OverQoS, the net aggregate rate looks like TCP. If they were just looking at the flow and appending 30% more packets, there would be a problem; however, they are concerned with the aggregate rate.

DISTRIBUTED HASH TABLES SESSION

Summarized by Magdalena Balazinska, Chip Killian, and Praveen Yalagandula

Designing a DHT for Low Latency and High Throughput

Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris, MIT Computer Science and Artificial Intelligence Laboratory

Frank Dabek presented the need for a DHT with low latency and high throughput by citing the infeasibility of realizing a backup system on a PlanetLab type environment, where bulk reads and writes experience a low throughput of 10Kbps and any operation suffers from a high latency of about 450ms.

This work has two main contributions. The first is a series of modifications made to the original Chord design that reduce the lookup latency. The second is a new transport protocol called STP, which makes use of per-hop ACKs and a sliding window for RPCs to achieve good throughput as well as good latency.

The first set of performance changes is the switch from iterative to recursive routing, using proximity neighbor selection, striping blocks across nodes, and using replication and neighbor selection to choose the closest nodes that also contain the interesting data. Low latency, however, is not sufficient, which is why they also consider high throughput.

To achieve high throughput, there is a need to efficiently manage the many connections needed by the parallel downloads. Opening many TCP connections is not efficient, because each one imposes a start-up latency (slow start), takes time to acquire a good congestion window estimate, and uses resources. In

contrast, using a few persistent connections constrains communication to the overlay links only. The fundamental problem is that there is a need to limit the number of parallel connections currently maintained (and for those connections to not use TCP). For this purpose, STP keeps a sliding window of RPC connections to control the number of outstanding calls. Combined, these two techniques can both decrease latency and increase throughput.

More information is available at *http://pdos.lcs.mit.edu/chord*.

Q: Amin Vahdat, UC San Diego. Regarding performance results of STP vs. TCP. Is it inherent that blocks have to be sent through overlay links when using TCP connections?
A: Yes, because you don't want to open connections to all nodes.
Q: Terence Kelly, HP. Are any methods from the Microsoft USITS paper on Skipnet applicable?
A: In Skipnets, nodes with the same geographic location have the same name, so lookups stay local when possible. No such change in namespace is needed in the Chord approach.
Q: Nick Murphy, Microsoft. What about insert and update traffic?
A: Coding is a good example of the tradeoff between read and write performance. All lookup optimizations would be the same.
Q: Nick Murphy, Microsoft. Are you storing to stable storage or in memory?
A: We use a database, so we do store to stable storage.
Q: Achim, UC Berkeley. Can we use DHTs as a backup store? How much data did you store in the system and per node? What about reliability?
A: The cost of keeping nodes in sync is future work.

Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays

Venugopalan Ramasubramanian, Emin Gun Sirer, Cornell University

Passive caching as done by current DHT algorithms suffers from two drawbacks: (1) The heavy tail of the zipf type access distributions implies that the caching cannot reap many benefits and (2) mutable objects give rise to coherency problems, and the caching can only provide a weak consistency model. Rama described Beehive, a general proactive replication framework for DHTs that automatically replicates and places the objects on several nodes in the DHT, satisfying the application-specified latency requirements while optimizing the maintenance bandwidth. Beehive achieves this through an analytical model based on a closed-form optimal solution that fulfills the application's requirements with the minimum number of replicas and that approximates that closed-form solution with nodes deciding locally how much to replicate objects based on estimated popularity.

Rama also presented CoDoNS, a cooperative domain name service, built on Beehive, to replace the traditional DNS. Through experiments on PlanetLab with MIT DNS traces as workload, the authors observed that Beehive (1) has very low latency (7ms vs. 39ms in legacy DNS), (2) is resilient against denial-of-service attacks, and (3) propagates updates quickly.

More information is at *http://www.cs.cornell.edu/people/egs/beehive*.

Q: Matt Welsh, Harvard: The goal of replication is failure protection, but you did not evaluate failures.
A: Yes, we did evaluate for some churn rate. The key point of the talk, however, is to decrease latency in lookup.
Q: Rob Szewczyk, UC Berkeley. Caching the Web would require too much space. Is this approach useful for large workloads?
A: Better than passive caching, but we didn't try large workloads.
Q: Mike Walfish, MIT. What happens when alpha is greater than 1?
A: Optimality factor is not valid for alpha greater than 1.

Efficient Routing for Peer-to-Peer Overlays

Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues, MIT Computer Science and Artificial Intelligence Laboratory

An important challenge in structured peer-to-peer systems such as Chord is to achieve efficient routing in spite of frequent changes in membership. Currently, most peer-to-peer systems perform lookups in time logarithmic to the size of the system, because nodes know only about log *N* other nodes. This large number of hops causes high latency.

The main thesis of the work that Anjali presented is that it is "possible to keep full routing state on every node and route in one overlay hop." Two approaches are proposed. The first scales up to order 100K nodes and routes in one hop. The second routes in two hops, but it scales to significantly larger networks (>1M).

To achieve one-hop routing, every node keeps a complete routing table. To keep these routing tables up-to-date, nodes organize themselves into a hierarchy over which all membership-change events propagate. The hierarchy is as follows. The ring is divided into a configurable number of slices. The node with an identifier closest to the middle of the slice is the slice leader. Each slice is subdivided into units. The node in the middle of the unit is the unit leader. All membership changes propagate up from nodes that detect the event to slice leaders. Periodically, slice leaders exchange information about changes in their

respective slices and push these changes down to nodes within their slice.

Slice leaders are the most loaded nodes because they send all membership updates to all other slice leaders. The load is low for systems less than a million nodes in size, but at greater sizes we have to either reduce the load or switch to a super-node scheme. One solution to alleviate the load is for slice leaders to send only membership updates about selected representative nodes within their slices. This approach improves scalability but increases lookup time by one hop since lookups now go through representative nodes before getting to the final destination. The first hop, however, can take advantage of proximity routing.

Q: Sean Rhea, UC Berkeley. Is the gain of the one-hop routing significant, since Frank Dabek argues that using proximity routing reduces latency to only two hops across the network?
A: This scheme has a latency of 1.5 hops across the network, and the first hop can benefit from proximity routing.
Q: Franklin, Northwestern. In the experiments, you assume that lifetime is exponentially distributed, but in real networks it appears that time is Pareto distributed instead.
A: Exponential distribution is more extreme, so the results would be even better if we assumed a Pareto distribution.
Q: Chunqiang Tang, University of Rochester. There are two types of traffic in the system: maintenance traffic and lookup traffic. Your scheme reduces lookup traffic but increases maintenance traffic. Shouldn't you be optimizing for the sum of the two instead?
A: Our goal is to reduce lookup latency, not traffic. Also, we expect lookup traffic to be significantly higher than maintenance traffic.
Q: Jonathan, University of Utah. What if slice leader crashes?
A: If a lookup fails, a node generates an event, and all routing tables will get updated so a new node will become the slice leader.

Stefan Savage, Co-Chair, presenting the best student paper award

SECURITY AND BUGS SESSION

Summarized by Vinay M. Igure and Chunqiang Tang

LISTEN AND WHISPER: SECURITY MECHANISMS FOR BGP

Lakshminarayanan Subramanian, Ion Stoica, Randy H. Katz, University of California, Berkeley; Scott Shenker, University of California, Berkeley and ICSI; Volker Roth, Fraunhofer Institute, Germany

This was awarded Best Student Paper. BGP is a path vector protocol that has policy-based routing. It has a control plane and a data plane, and invalid routes in either of these planes could cause widespread damage across the Internet. In the control plane, a router can propagate spurious routes, and in the data plane, routers may inconsistently forward packets. Some of the possible attacks could be black hole attacks, impersonation, and eavesdropping. Lakshmi also provided some real-world examples of attacks that targeted the BGP. In order to deal with these threats, this paper presented a combination of two mechanisms called Listen and Whisper. The approach uses Whisper to deal with the attack at the control plane and Listen to deal with it at the data plane.

Lakshmi compared this new approach with other previously proposed mechanisms, such as a key-based mechanism, centralized databases, configuration-checking tools, and data-plane route-probing tools. Verification using PKI can pinpoint the source of the problem, but this is not possible with Listen and Whisper. Whisper is based on the Chinese whisper game, which has two versions: split-whisper and loop-whisper. A demo of the game was given to easily explain the Whisper route consistency checking mechanism. Due to time limitations, only a brief summary of Listen was presented. The basic idea behind Listen is that if TCP data contains a SYN and DATA packet, it implies that an ACK has been previously received and a valid route exists. The biggest challenge with this approach was to deal with the false positives and false negatives. Listen reduces both of these to less than 1%.

Although these two techniques can be used individually, they offer best performance when used jointly and the results of the two mechanisms are correlated. Lakshmi presented the results of testing the effects of colluding adversaries on the current Internet and compared it with the effects on Whisper implementations. It shows that Whisper policies reduce the percentage of affected ASes.

Measurement and Analysis of Spyware in a University Environment

Stefan Saroiu, Steven D. Gribble, and Henry M. Levy, University of Washington

Steven Gribble presented the results of the study done at the University of Washington to investigate the problem of spyware. The popular media coverage of spyware generally scares people. Politicians have also stepped into the field and are trying to create laws to deal with this problem. Given the privacy and security risks, it is essential to conduct rigorous research into this problem. The biggest problem is defining spyware. For the purpose of this study, spyware was defined as any software that collects personal information and relays it to a third party.

The work focused on studying the prevalence of four major spyware products: Gator, Cydoor, eZula, and SaveNow. The authors created network signatures for each of these by monitoring the HTTP traffic. The university network traffic was sniffed for seven days and then analyzed to find these signatures. It was found that 5.1% of machines were infected with spyware. There were many interesting correlations, but not necessarily causations, between the presence of spyware and the number of Web servers contacted, Web objects downloaded, executables downloaded, and presence of P2P file sharing software. Most P2P software comes with spyware. The study also found that of those PCs infected with spyware, many contained more than one spyware program and most spyware lingered in the machines for extended periods. The work also found security flaws in the "auto-update" feature of eZula and Gator that could be exploited to launch attacks. Given the widespread deployment of spyware, an attack exploiting these bugs could cause enormous damage.

Steven also noted that it is difficult to implement technical solutions to deal with spyware, primarily because most users choose to install these programs and because it is very hard to distinguish between spyware and non-spyware programs. Legal solutions are also difficult for these reasons. However, detecting the presence of spyware is easy and hence the focus should be on containment and removal.

Model Checking Large Network Protocol Implementations

Madanlal Musuvathi, Dawson R. Engler, Stanford University

Madan introduced the general problems associated with checking network protocols. Conventional testing of protocols is usually inadequate. The approach taken in this paper is to model check the entire protocol implementation. Model checking explores all possible states of the protocol and checks for vulnerabilities at each state. Since the state space is usually large, the checking is done until all the resources are exhausted. Madan used CMC, a C Model Checker, that checks code directly, to test protocols. CMC is similar to a network simulator but provides the ability to checkpoint implementation states. CMC also computes the signature of each state, which can then be compared to determine if they are similar.

The first test of CMC was done on AODV, a routing protocol for mobile ad hoc networks, and they were successful at finding bugs in the implementation code. The next step was to scale CMC to test a large protocol such as the TCP protocol in a Linux implementation. TCP was chosen because it is a hard protocol and widely tested. Madan highlighted the contributions of the work, which included demonstrating that CMC is applicable for TCP and that the techniques used to check TCP can be used to scale CMC to test other large systems. The work also provides a general infrastructure that can be used for testing large systems. The biggest problem in testing TCP was in isolating the TCP code from the Linux kernel. Since that was not possible, the entire kernel was ported into CMC with well-defined interfaces to run TCP code.

The techniques that were used to deal with exploring large state spaces – handling large states and incremental state transitions – were discussed. The boot-up state was the initial state, with single-step transitions between states. Since the states were large and transitions slow, a "copy-on-write" mechanism was used. In order to deal with the large states and their increments, states were split into objects and made to share unmodified objects. To handle heap canonicalization, Madan presented an incremental heap canonicalization algorithm, discussing the performance improvements achieved. The CMC checked for generic correctness properties such as memory leaks and for protocol conformance. The TCP RFC was translated to C. They found four bugs in the code. Some of the future work includes using this approach to test the entire Linux kernel.

RESOURCE MANAGEMENT SESSION

Summarized by Laura Grit and Piyush Shivam

Constructing Services with Interposable Virtual Hardware

Andrew Whitaker, Richard S. Cox, Marianne Shaw, and Steven D. Gribble, University of Washington

Andrew Whitaker presented mu-Denali, an extensible design for constructing virtual machine (VM) services. The authors argue that although VMs are powerful platforms for allowing services, developing services is poor since VMs are closed systems and are difficult to tinker with. They believe that the correct VM service development platform is one with extensible design and a clean pro-

gramming API. Virtual machine monitors have a "whole-system perspective," which makes possible a variety of interesting services like migration, intrusion detection, and replay logging. The key ideas in their mu-Denali architecture are the interposition of events generated in one VM's virtual hardware device by another, the extension of the virtual architecture of the VM, and, finally, a clean programmable API that lets one build the VM services.

The experimental evaluation of the system addressed the overhead of interposition and mu-Denali's ability to support the development of VM services. It was observed that system call and packet-intensive operations suffer degradation in performance because of extra trapping, copying, and so on. However, the overall application throughput numbers were acceptable and the development effort minimal.

There were questions related to Apache migration, its network connections, and the benefits of restarting VMs instead of application. Andrew acknowledged the limitation of the LAN-based implementation (no IP routing) and mentioned that restarting VMs was a part of completeness argument. Jay from Sun Microsystems pointed to some related work in this area, to which Andrew acknowledged that a lot of interposition work is existing in the same space with some differences. The mu-Denali approach gains a lot by using a generic OS. Someone asked if anyone outside of the authors' group had tried to build the services using the primitives, to which Andrew replied no.

SWAP: A Scheduler with Automatic Process Dependency Detection

Haoqiang Zheng and Jason Nieh, Columbia University

Haoqiang stated the motivation for his work by emphasizing the large number of dependencies that exist between the many components in creating tasks. Existing solutions have limitations since they do not address the dependencies not caused by mutexes such as signals, dynamic priorities, or priority inheritance.

Their solution, SWAP, is a simple model that detects dependencies based on resource access history and uses a feedback loop to improve accuracy. In their model, which is transparent to users, everything is treated as a resource (e.g., sockets, file locks, and semaphores) and all requests use system calls, which they use to see what resources have been requested. Past resource providers are predicted as potential current resource providers. Each provider is assigned a confidence level based on system feedback of how they have performed as a resource provider in the past.

The SWAP system uses existing schedulers as black boxes. After the existing scheduler runs, they use the SWAP scheduler with dependency detection to determine what was scheduled. Their system was implemented with a minimal kernel-level change and by adding dependency detection to the system call layer.

In their experiments, they showed how high-priority tasks would benefit from SWAP. When system load was low, SWAP had no effect and did not impose any system overheads. However, when system load was high, SWAP performed much better and increased throughput compared to that in naive Linux.

In response to Steve Gribble's question whether they can catch all dependencies, Haoqiang said that they found all system calls on Linux. For more information, see *http://www.ncl.cs.columbia.edu*.

Contract-Based Load Management in Federated Distributed Systems

Magdalena Balazinska, Hari Balakrishnan, and Mike Stonebraker, MIT Computer Science and Artificial Intelligence Laboratory

Magdalena presented her load management system to provide mechanisms for handling peak system load without overprovisioning and with wide area distribution of load in such federated systems. The system tries to achieve the challenges of having sufficient incentives to participate, efficiency of mechanisms, and customization options for participants. In this system, pair-wise contracts are negotiated between participants offline to specify a fixed price per unit of load. The system goal is an acceptable (not optimal) allocation of resources: Participants compute marginal costs for current loads and compare this to contract prices to determine if it is better to transfer or keep load. The fixed prices are chosen around the gradient of the cost function. Magdalena pointed out that fixed price constraints can cause unacceptable allocations, and this can be rectified by using a small price range, which allows load to propagate through a chain of identical contracts. Moreover, since the price contracts are done offline, they can be greatly customized, which enables service discrimination. The implementation of these mechanisms was presented in Medusa to show that this approach is simple, feasible, efficient, and customizable.

When asked how this approach can gain widespread use, she responded that the system is user-friendly and that by having contracts offline, payments can be specifically customized to fit the participant's needs. Simulations included a thousand participants, each with only a few contracts, and they found acceptable allocations, so Magdalena felt it was a scalable approach. She also felt that although oscillations of load were rare,

this technique was designed to deal with occasional load spikes. For more information, see *http://nms.csail.mit.edu/projects/medusa*.

DHT APPLICATIONS

Summarized by Magdalena Balazinska, Chip Killian, and Praveen Yalagandula

Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval

Chunqiang Tang and Sandhya Dwarkadas, University of Rochester

This paper works to solve the problem of full-text search in peer-to-peer networks, in contrast to the traditional keyword search. The approach is to leverage DHTs to help deal with the information explosion problem. This is done by creating inverted lists by keyword and storing both the inverted lists and the complete word lists on each node that the keyword maps to.

By storing the index in this fashion, you can locate any document by keyword by searching a single node in the DHT. And if you are performing a query that deals with more than a single-word search, you can still perform the query with just one computer, because you can use the word list to complete the query.

To improve upon these results, however, and to reduce the storage cost, they further focused on weighted keywords. In this fashion, they extracted the top X keywords from each document, where added weight is given to keywords that appear frequently in this document, but seldom in other documents. Two other optimizations include rebalancing the load in the system by mapping a keyword to a range of hash addresses instead of a single address, and using automatic query expansion, which maps terms to other likely interesting candidates to return additional relevant pages.

Q: David Oppenheimer, UC Berkeley. New nodes join a system at appropriate places in the ID space. Why did you make this assumption? How often do you assume that nodes are joining?
A: We only assume a computer joins once.
Q: Matt Welsh, Harvard. Why does your approach require so much space per node compared to Google?
A: We replicate word lists, too. Compared to Google, we don't have to search every computer.
Q: L., UC Berkeley. What about ranking query results?
A: We only search for computers responsible for keywords.

Untangling the Web from DNS

Michael Walfish, Hari Balakrishnan, MIT Computer Science and Artificial Intelligence Laboratory; Scott Shenker, ICSI

This paper had two fundamental tenets. First, that we should restructure the Web such that Web links contain no identifiers as to whom they can be associated with or where they might be located. This would then make links stable, and when they change physical location or ownership, their URL would not change. The second tenet for the talk was the design of an appropriate lookup service based on the non-identifying IDs.

The first part of the talk argued that Web links as currently implemented suffer from problems when changing locations or owners, from the point of view of technical challenges as well as the idea of vanity domains. The main conclusion is that, no matter what the design of user-friendly-identifiers is, humans will always fight over them, making these systems unsuitable for stable URLs. Instead, by using a flat-space identifier with an update scheme, these two identifiers can be completely decoupled. By doing so, we get: (1) stability, where references remain invariant when objects change; (2) support for object replication, where one identifier can map to a list of replicas; and (3) automatic management of namespace.

The design in the second part of the talk was rather straightforward. The authors used a DHT. To allow updates and manage changes safely, the authors used self-certifying records.

Q: Doug, University of Wisconsin. If I establish a DNS server, it is my responsibility to serve the names. With your scheme, everyone is responsible to serve names.
A: We envision a management model for the infrastructure, but it could be at the level of the individual doing puts and gets.
Q: Jonathan, University of Utah. One way DNS is used is to locally resolve suffixes, and no one knows about these in the outside world.
A: Details about write locality are in the paper.

Democratizing Content Publication with Coral

Michael Freedman, Eric Freudenthal, and David Mazières, New York University

It is well known that flash crowds bring down small Web sites. Existing approaches that deal with flash crowds are too expensive for these small Web sites (content distribution networks, load balanced servers, etc.), while client-side proxying does not provide 100% coverage. Instead, CoralCDN proposes to use volunteers to pool resources and dissipate flash crowds.

CoralCDN is a decentralized, self-organizing, peer-to-peer Web-content distribution network. To use CoralCDN, a content provider must rewrite its URLs into "Coralized" URLs (e.g., *www.x.com* becomes *www.x.com.nyud.net:8090*). This directs unmodified browsers to Coral, which absorbs load.

To provide such service, CoralCDN has two key components. First, Coral DNS servers map clients to nearby Web proxies, by probing clients to determine any nearby nodes, based on either network

topology hints or proximity measurements. Second, Coral Web proxies enable clients to retrieve a locally cached copy of the data or find a nearby neighbor that has the data, without needing to contact the origin Web server.

To enable the above functionality, Coral provides the abstraction of a distributed index built on a key-based routing layer. Coral nodes organize themselves into a hierarchy of clusters based on RTT distances between nodes, so that requests remain local when possible in order to minimize latency. Values are stored once in each level cluster. To always store a value at the closest node in the identifier space, however, would cause hotspots. Instead, Coral introduces a rate-limiting mechanism, by halting the publishing ("put") operation as soon as it finds a full and loaded node.

More information is available at *http://www.scs.cs.nyu.edu/coral*.

Q: Student from UC San Diego. What would be the motivation for people to install Coral?
A There are already volunteers that mirror Slashdot. Organizations can benefit by improving local client latency, reducing downstream bandwidth usage, and offering locally trusted proxies to provide security benefits to their clients.
Q: Steve, University of Washington. What are possible abuses of Coral?
A: There are several challenges. One is that nodes could overwhelm each other with requests. We could employ rate-limiting mechanisms at the application layer. Additionally, a node could donate only upstream bandwidth to prevent distant clients from replacing their local cache. Lastly, there are integrity concerns, but we could leverage content-hashes in URLs that can be verified by the client's proxy, which is the incentive for running local proxies.
Q: Rama, Cornell. What is the latency you get when a flash crowd occurs?
A: Please refer to the graph in the paper. Median latency is approximately 100ms for a hierarchical system on PlanetLab.

OVERLAY NETWORKS SESSION

Summarized by Suchita Kaundin and Aydan Yumerefendi

Operating System Support for Planetary Scale Network Services

Andy Bavier, Scott Karlin, Steve Muir, Larry Peterson, Tammo Spalink, and Mike Wawrzoniak, Princeton University; Mic Bowman, Brent Chun, and Timothy Roscoe, Intel Research; David Culler, University of California, Berkeley

Timothy Roscoe described the main design principles and organization of PlanetLab – an open, globally distributed platform for developing, deploying, and accessing planetary-scale network services. The challenge behind PlanetLab is to build a new distributed system with the existing technologies, which should be able to evolve under the simultaneous development of a large research community. The main design goal is to implement a minimal set of abstractions and interfaces to enable service builders to construct powerful applications, yet protect and isolate competing and co-existing services. PlanetLab also supports unbundled management – independent evolution of its abstractions from the operating system running on each node.

PlanetLab consists of a collection of geographically distributed physical nodes. Each node can host a number of virtual machines (VM) that interface with the underlying hardware using a common VM monitor. A set of distributed VMs forms a slice, which is the basic unit of resource allocation and isolation; each service in PlanetLab runs within a slice. Within each system node, two important VMs provide a minimal set of administrative privileges, the node manager manages and monitors the VMs running on the node, and the local manager is a privileged VM that allocates local resources to a VM.

PlanetLab is a work in progress, and only time will tell which infrastructure will evolve to give it fuller definition. The design allows network services to run in a slice of PlanetLab global resources with the PlanetLab OS, providing only local abstractions and as much global functionality as possible. In the future, its designers expect to limit the functionality implemented in privileged services and allow service builders to implement as much low-level functionality as possible.

MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks

Adolfo Rodriguez, Charles Killian, Sooraj Bhat, and Dejan Kostic, Duke University; Amin Vahdat, University of California, San Diego

MACEDON is an infrastructure for building, deploying, and evaluating large-scale distributed systems. It automates a large number of commonly performed routine tasks in the process of developing and evaluating large-scale systems. MACEDON is general, and shows high performance, and its building blocks can be easily replaced. For novices, the infrastructure is easy to use, yet it does not limit experts from customizing the system to their specific needs.

The main idea of MACEDON is to separate specification from implementation. To develop a system using MACEDON, system developers use a C++-like domain-specific language to define the specific algorithm used in their system and make use of the API exported by MACEDON to automatically perform tasks related to failure detection, timers, transports, bloom filters, etc. MACEDON parses the specification and translates it into executable code that uses library functions and the MACEDON

code engine. The engine and code libraries are common to all systems and thereby increase evaluation consistency and code reuse. The final code can be either emulated or run live on the Internet.

MACEDON uses a finite state machine approach to represent the state of each node in a distributed system. In this model, events such as message reception, scheduled completion of timers, and application commands trigger actions. Actions include setting local node state, transmitting new messages, scheduling timers, and delivering data. In addition, events may cause the protocol to move from one system state to another.

MACEDON's approach has been validated by implementing a large number of overlay protocols and comparing their performance to the official published results. Each implementation is smaller than 600 lines of MACEDON code and shows performance comparable to its original custom implementation containing thousands of lines. One question left unanswered by the current implementation is about abstraction fairness, i.e., whether some algorithms get better treatment than others.

Structure Management for Scalable Overlay Service Construction

Kai Shen, University of Rochester

Kai Shen described Saxons, a service-independent distributed software layer that dynamically maintains a selected set of overlay links for a group of nodes. Saxons is designed to provide efficient overlay connectivity support that can assist the construction of large-scale Internet overlay services. This new software layer maintains high-quality overlay structures with three performance objectives: low path latency, low hop-count distance, and high path bandwidth. At the same time, it is scalable, introduces low overhead, and demonstrates stable behavior.

Saxons achieves its design goals by controlling the management overhead introduced by probing and maintaining the structure. In this system the per-node management cost depends only on the number of attached links, not on the overlay size. Saxons uses a randomized membership protocol to avoid maintaining and exchanging complete membership information. Randomization is used to maintain and exchange membership information as well as to discover nearby hosts. Each node in the overlay has a limit on the number of overlay links it can establish and accept.

To maintain the quality of the structure, Saxons can make use of three algorithms that either minimize latency, minimize latency on half of the links or choose the remaining half at random, and minimize latency to half and choose the remaining half at random from nodes with high bandwidth. To avoid structural oscillation, Saxons performs link adjustment only if the new topology persists for a period longer than a given threshold.

Saxons was evaluated using simulation and live PlanetLab deployment. The simulations and experiments on 55 PlanetLab sites demonstrate Saxons' structural quality and the performance of Saxons-based service construction. It is believed that it is more feasible for sharing low-level activities such as link property measurements. Further investigation on this issue is needed.

RELIABILITY SESSION

Summarized by Aydan Yumerefendi and Suchita Kaundin

Session State: Beyond Soft State

Benjamin C. Ling, Emre Kiciman, and Armando Fox, Stanford University

Session state is an essential part of any dynamic Web application: It stores important per client information for the duration of a client's session. Benjamin presented SSM, a system that decouples the management of session state into a separate dynamic layer. As a result of its design, it decreases management overhead and provides efficient recovery and load balancing.

SSM makes use of a secondary storage hashtable. The table consists of a number of bricks, which store per-client session data. Web servers use a client-side library to communicate with the storage bricks. Writes in SSM avoid two-phase commit by selecting a random subset of bricks to write to and waiting until only a fraction of them acknowledge the write. SSM does not make use of an index to locate data. Instead, the Web server prepares an HTTP cookie with information about the location of the client's session state and sends it back to the client. When reading data, the Web server makes use of this cookie to request information from the specified bricks.

SSM tolerates failures gracefully: it preserves the availability of data during failure and recovery. Brick recovery is achieved for free by restarting the brick. Client usage patterns cause data to be rewritten and ensure sufficient availability even in the presence of failures. SSM makes use of admission control to balance the load. Failure detection in SSM is based on Pinpoint, a framework for detecting likely failures. Pinpoint monitors a set of brick parameters and considers as failed any brick with parameters that deviate from the statistical averages.

The evaluation of SSM shows that it preserves throughput, improves latency, and enforces high availability. Benjamin also mentioned that there has been interest in SSM from the industry.

Path-Based Failure and Evolution Management

Mike Y. Chen, Dave Patterson, and Eric Brewer, University of California, Berkeley; Anthony Accardi, Tellme; Emre Kiciman, Armando Fox, Stanford University

Chen started by emphasizing the importance of fast recovery and rapid online evolution. Fast recovery is important in preserving availability, and rapid online evolution is necessary for the continuous software updates of large-scale systems. To facilitate both tasks, Mike presented an approach that monitors the path of each request in a system, aggregates information, analyzes the behavior of individual components, and quickly detects faulty nodes and recovers them to a stable state.

The key idea of this approach is to separate observation from analysis and use statistical techniques to achieve its goals. Each request in the system carries a unique identifier and a timestamp. The system records the components that the request visits, aggregates the different observations, and tries to draw statistical conclusions using machine learning techniques. The C4.5 decision tree learning algorithm used in the project can tolerate noisy data and helps reliably identify problematic components and system states. It also supports application evolution management by tracking component dependencies.

There are three path-based macro-analysis implementations of the above ideas: Pinpoint, ObsLogs (used by Tellme Networks), and SuperCAL (used by eBay). They all show good performance and help achieve fast recovery and rapid evolution management. The total overhead of monitoring and analysis is within 1 to 3% on tests performed on eBay. The future plans for the project involve extending its approach to wide area systems and multiple administrative domains.

Consistent and Automatic Replica Regeneration

Haifeng Yu, Intel Research Pittsburgh and Carnegie Mellon University; Amin Vahdat, University of California, San Diego

Replication improves availability of large-scale systems but also presents significant management overhead due to the time and effort spent to preserve the consistency of a replicated system. Haifeng Yu presented Om, the first peer-to-peer wide-area read/write storage system that improves the manageability of large-scale distributed systems through online automated replica regeneration while preserving system consistency. These properties are achieved through three novel techniques: (1) single-replica regeneration helps achieve high availability with a small number of replicas; (2) failure-free reconfigurations allow fast reconfiguration within a single round; and (3) a lease graph and two-phase write protocol serve to avoid expensive consensus.

Om uses Pastry, a distributed hashtable, to construct an overlay network and route messages. Om stores each data object on a number of replica nodes. Reads for an object go to any of the replicas, while writes involve the whole replica group. When failures occur, it is important that all nodes have the same system configuration. Instead of using distributed majority quorum to achieve this goal, Om uses the witness model. The witness model allows quorums as small as a single node, by choosing random system nodes and using their view of the system and the intuitive fact that the different views do not diverge significantly. Om organizes witnesses into a matrix, and each replica tries to coordinate with one witness from each row. The witness model requires at least one common witness for a replica group to achieve unique configuration. Multiple rounds can help address the problem of non-intersection; the expected number of such rounds is 3.1.

The evaluation of Om involved LAN emulation and live PlanetLab deployment. In all experiments the system showed high performance and availability. Regeneration completes in approximately 20 seconds, availability can be as high as 99.9999% with only four replicas, and the probability of incorrect configuration is 0.000001 and is independent of system size.

STORAGE SYSTEMS SESSION

Summarized by Piyush Shivam and Ramakrishna Kotla

Total Recall: System Support for Automated Availability Management

Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker, University of California, San Diego

Ranjita began by presenting the definition of availability and indicating that redundancy provides availability. She then outlined two fundamental questions: (1) How can one quantify the availability of systems? (2) What is the exact relationship between redundancy and availability? Total Recall tries to address these questions by providing redundancy management knobs for automatic maintenance of availability metrics. This is done using mechanisms such as availability prediction, redundancy management, and dynamic repair to meet required availability goals.

Ranjita went on to describe in more detail what these mechanisms are and how they interact with each other. Given the target levels of availability and current system availability, a policy module

decides whether to use replication or erasure coding for redundancy and whether to use eager or lazy repair for dynamic repair. This information is then fed to the redundancy management module, which predicts the degree of redundancy as a function of the availability, coding, and repair scheme. Ranjita then gave an experimental evaluation of the system for which she used a prototype implementation incorporating the ideas outlined earlier.

During the discussion, someone asked if the availability of the master node is higher than other nodes. Ranjita replied that availability of all nodes is the same and if the master fails, another node takes over as master. Concerns over the impact of extra redundancy over consistency were raised, and she acknowledged that as an important issue and mentioned it as something for future work. She was asked why the bandwidth numbers in the graph were so high and replied that it is because of the way the simulation was done: they are just the upper bounds; the actual bandwidth is much less. Finally, she was asked if it really makes sense to store data on unreliable components. She answered yes, it does make sense by using the mechanisms outlined in the talk. For further information, please refer to *http://ramp.ucsd.edu/projects/recall/*.

TimeLine: A High Performance Archive for a Distributed Object Store

Chuang-Hue Moh and Barbara Liskov, MIT Computer Science and Artificial Intelligence Laboratory

Barbara began by introducing the idea of a timeline service and what it does. The key idea is to take on-demand snapshots of data at a user-specified time and run computations on data sometime in past. This enables time travel for computation. However, their systems allow read-only computations on the snapshots and do not allow modifications to the state stored in the snapshot.

This system is built in a client-server setup using the Thor environment, which provides support for persistent object store. The users run computations at the client machines, and the persistent state resides at the servers.

She described several approaches for building such a system, with their problems, and then described the TimeLine approach, in which each snapshot has a time which tells what modifications have been made up to that point. The association of time with the modification is done using the idea of sender time, receiver time and Lamport's logical clocks. Finally, she presented the implementation of this system, which met the goals of not disrupting the rest of system while taking snapshots and of avoiding penalty for taking snapshots.

In the question-answer session, someone asked if it is possible to provide read and write computations on the snapshots. Barbara replied that it is possible but one will get many versions of system state at a given time and the overall system will become very complex. The second question was whether clients have to participate in the snapshots, to which Barbara replied yes, and mentioned that in general things need to move through the client. Another question was what happens to writes in the system when Thor gets replaced by a general file server? Barbara replied that the kind of optimization with respect to writes (e.g., delayed writes or write gathering) depends on the semantics of storage system and file system. The current implementation takes advantages of semantics provided by Thor.

Explicit Control in the Batch-Aware Distributed File System

John Bent, Douglas Thain, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Miron Livny, University of Wisconsin, Madison

John Bent began by motivating the central question behind this work – harnessing remote storage for batch workloads. The key questions addressed in this work are: (1) Do the batch computing workloads offer new challenges to the distributed computing infrastructure? (2) Is the current infrastructure okay for such a need, and, if not, what is the appropriate architecture? Having motivated the work, he went on to describe the batch computing environment, batch workloads, and the I/O characteristics of the same. In such environments the key challenge is to manage the flow of data into, within, and out of the system. The current distributed file systems are not practical for such applications, because they make uninformed decisions about caching, consistency, and replication. On the other hand, the Batch-Aware Distributed File System (BAD-FS) exposes this knowledge and removes the guesswork. In addition, it is practical and deployable, because it is packaged as a generic batch system.

BAD-FS makes intelligent scheduling decisions by using the remote cluster knowledge of storage availability and failure rates, and workload knowledge regarding data type, data quality, and job dependencies. John went on to explain the performance enhancement techniques used by BAD-FS, like I/O scoping (eliminating unnecessary I/O) and capacity-aware scheduling, which makes the system appealing for batch workloads. He went on to describe a simplified implementation of this system and the scientific workloads that were analyzed/run under BAD-FS.

In the discussion, someone asked if BAD-FS is burdening the user to find

out the extra knowledge which the BAD-FS scheduler uses to make intelligent decisions. John replied that the ultimate goal is to automate this process, and in fact the BAD-FS architecture reduces user burden for organizing scattered data. Another question was whether the intermediate data between different stages of a job needs to be maintained, since it is uncommonly an input to the next stage. John answered that they have not looked at that, but there are certain applications that overwrite the output data, and hence it becomes important to maintain it. For more information on this work, please refer to *http://www.cs.wisc.edu/adsl*.