conference reports

This issue's reports focus on WIESS '02 and on OSDI '02.

OUR THANKS TO THE SUMMARIZERS:

Scott Banachowski
Richard S. Cox
Steven Czerwinski
Himanshu Raj
Cristian Tapus
Charles P. Wright
Praveen Yalagandula
Wanghong Yuan
Nickolai Zeldovich
Ben Zhao
Yutao Zhong

2nd Workshop on Industrial Experiences with Systems Software (WIESS '02)

BOSTON, MASSACHUSSETS
DECEMBER 9-11, 2002

KEYNOTE ADDRESS

Douglass J. Wilson, IBM

Summarized by Richard S. Cox

MIT's *Technology Review* recently ran a story titled "Why Software Is So Bad."

The key is the problem of integration. CIOs spend 35% of their budgets on integration, because every new system must work with the existing infrastructure. The complexity of integration is driven up by the constraints of the business environment as well as those of the software.

Several lessons can be learned from studying systems usage. First, standards and componentization are proving ineffectual for complex systems. For example, LDAP is a fine protocol, but no two organizations use the same schema. Making matters worse, interoperability is poor due to differing interpretations of standards, edge conditions, and vendor-specific extensions. This is leading to a change from creating solutions by mixing "best-of-breed" products to using a single "best-of-suite" package. Unfortunately, much of the literature on building component systems is academic, failing to deal with the scale of large systems.

Second, systems will fail. Other industries have accepted this, but software engineers are just now realizing that failure is hard. The recovery design must fit the usage, which means the designer must understand the failure modes in practice. This may mean using less sophisticated algorithms that are better fitted to the purpose. It also means accepting that business redundancy may be at odds with IT redundancy. For example, a stock trading system may have several redundant pathways for entering a trade, to protect against trades being lost before they have been entered. For the IT infrastructure, this means the redundant pathways need to be synchronized at some point. This type of problem is rarely considered by researchers or product developers.

Third, error logging and reporting is important. As an industry, we currently support very primitive logging with no mechanisms for root-cause analysis or correlation of failures. Error messages are often arcane or not useful, and "first-failure" capture is impossible. This is evidenced by a common, though unrealistic, request from support center staff: "Turn logging on and recreate the failure." Because logging events need to be correlated, error tracking and logging should be a basic service of the OS.

SESSION 2

Summarized by Wanghong Yuan

USING END-USER LATENCY TO MANAGE INTERNET INFRASTRUCTURE

Bradley Chen, Michael Perkowitz, Appliant

The problem addressed in this paper is that distributed application performance is important but hard to understand. CDN selection and CRM systems were offered as examples to illustrate the problem. The basic approach proposed is to use end-user latency analysis: (1) content (e.g., an HTML Web page) is tagged to collect data; (2) tagged data is observed on the desktop (end-client system); and (3) data is analyzed on the management server.

The challenges for this approach include (1) technique issues such as larger data sets, heavy-tailed data, and the derivation of request properties, and (2) social and economic issues such as privacy. The results show that end-user latency analysis can monitor relevant information, which is obscured otherwise.

Building an "Impossible" Verifier on a Java Card

Damien Deville, Gilles Grimaud, Université de Science et Technologies de Lille

The smart card device environment imposes constraints on CPU, memory, and I/O. As a result, Java Card Virtual Machine needs to be adapted to the smart card. The regular verification approaches do not fit, since unification is costly. The proposed approach addresses the above problems via (1) non-stressing encoding and (2) efficient fixed points using a software cache policy.

Enhancements for Hyper-Threading Technology in the Operating System: Seeking the Optimal Scheduling

Jun Nakajima, Venkatesh Pallipadi, Intel

In this talk, Jun Nakajima first gave an overview of Hyper-Threading (HT) technology by comparing it with multiprocessors. The reason behind HT is that CPU units are not fully utilized. To fully utilize CPU units, the HT approach is to use two architectural sets, thereby executing two tasks simultaneously.

The HT approach requires the OS scheduler to support HT-aware idle handling, processor-cache affinity, and scalability (per-package run queue). This paper proposes a micro-architecture scheduling assist (MASA) methodology to address the above problems, thereby achieving an optimal process placement.

INVITED TALK

Software Strategy from the "1980 Time Capsule"

John R. Mashey

Summarized by Yutao Zhong

John Mashey reused the slides from a talk he gave 25 years ago titled "Small Is Beautiful and Other Thoughts on Programming Strategies." It is interesting to see from these old slides and the newly added comments what has changed and what hasn't.

The previous talk was given in 1977, when the main computer models were IBM mainframes, coming VAX, and PDP-11s, while C was taking the place of ASM and structured programming became the dominating idea. Three approaches of building system software were introduced and compared: "Do it right," "Do it over," and "Do it small, with tools." "Do it right" emphasizes an optimistic on-requirement analysis that assumes "we know what we are doing." "Do it over" puts more emphasis on early implementation by still starts from scratch. The last approach, by contrast, considers tools instead of systems and builds small and fast so that, if necessary, failures can happen quickly.

In order to see the effect of these strategies, Mashey discussed different metrics to qualitatively measure success and gave statistics and observations of projects in data processing. Figures and numbers showed the low percentage of complete success and indicated the larger a project is, the higher overhead it has to pay. Laws of program evolution also state that the entropy of a project increases with time and may result in a complex program used to solve a simple problem.

Several principles were offered to counteract these problems: "build it fast," "keep it small and simple," and "build for change." Existing tools should be utilized whenever possible. It would be good to build tools and consider the interfaces of connecting tools. Some "small tactics," including "lifeboat theory," "sinking lifeboat theory," and other considerations about people and consolidations, were also discussed.

Even after 25 years of work, we need to keep these problems in mind, since system complexity is much higher nowadays; fortunately, people are increasingly aware of these issues.

Mashey ended the talk by saying, "We have met the enemy and they are us."

SESSION 4

Summarized by Cristian Tapus

An Examination of the Transition of the Arjuna Distributed Transaction Processing Software from Research to Products

M.C. Little, HP–Arjuna Labs; S.K. Shrivastava, Newcastle University

Arjuna started in 1986 as a research project at the University of Newcastle, England. Arjuna was a "vehicle for getting Ph.D. degrees." The decision to use C++ was a pragmatic one (expensive Euclid vs. free C++ AT&T). Arjuna was designed to be a toolkit for development of fault-tolerant applications which would provide persistence, concurrency control, and replication. Modularity was the key to the longevity of the system.

In 1994 Newcastle University asked them to implement a student registration system because the "academic researchers are cheap." The system was supposed to run on multiple platforms, serve about 15,000 students over five days, and could not tolerate failures. There were problems, though. Assumptions were made about network partitions and recovery that made the system fail to identify dead machines vs. slow connections. Intuition is not a good approach to designing systems.

The year 1995 brought standards for transactions: object transaction system specifications (OTSS) from OGM. It shared many similarities with Arjuna, but it was only a two-phase commit protocol engine (persistence and concurrent control where required from elsewhere). At this time the OTSArjuna system was developed. With only slight changes to the interfaces between modules, the system was complying with OTS. JTSArjuna followed just two years later as the first Java transaction service.

In 1999 the Java and C++ transaction service were marketed; only one year later Bluestone took over Arjuna Solutions Limited and was, in turn, acquired by HP in 2001. When the system was acquired by Bluestone the need for real testing became a reality. For the previous decade only about 20 tests had been used, but this was increased to over 4000 tests in order to stretch every feature of the system. The previous method was to get a release out to the users, and users would then report problems back and bugs would be fixed. Not anymore. The industry method was different from many perspectives: write manuals and white papers and train other people. "I used to laugh at white papers, but I realized you need skills. An academic person cannot do it. Academic people do technical reports, which are different," Little said.

Was it worth it? YES. But it was stressful moving away from R&D. "If you have a family don't do it. If you are in the industry and you feel you are stressed up, move to academia."

When asked what they would do differently, the reply was that they would (1) get somebody else to do the failure recovery and (2) make sure that they would have more than 20 tests.

Tree Houses and Real Houses: Research and Commercial Software

Susan LoVerso and Margo Seltzer, Sleepycat Software

Susan talked about the process they followed to make a commercial product out of BerkeleyDB. The main argument was that a research prototype is like a tree house – it doesn't last – while a commercial product is the real house. Sleepycat was founded in 1996 and transformed DB1.85 into a real product. It added transactions, utilities, and recoverability while continuing to be open source. Sleepycat is a "distributed" company; with employees spread across the world, it is hard for them to interact with each other directly. But the heterogeneous environment makes the company more powerful. In order to produce quality software, however, Sleepycat must follow rigorous software practices. Designs and reviews are sent to the entire engineering staff (one advantage of being small), and there are strict coding standards (it is the *law*).

The talk continued by describing techniques used to obtain the final product. When you hit bedrock, try to rethink what you are doing; and observe the "rule of holes – if you are in one, stop digging." In the end, certain lessons were learned from the development process: Designs and reviews are important, but reviews are not perfect; there needs to be a willingness to stop and change course when necessary and to throw code away, even if it works; and you need someone nearby who's close to the process but objective.

JOINT WIESS/OSDI PANEL

Research Meets Industry

Chair: Noah Mendelsohn; Panelists: Ramon Caceres, ShieldIP; Mark Day, Cisco Systems; Charles Leiserson, MIT; Dick Flower, HP; Brian Bershad, University of Washington

Summarized by Nickolai Zeldovich

The joint panel discussed issues related to the ridge between research and industrial development and their correlation. The discussion was entitled "Research Meets Industry."

Brian Bershad: Knowing how to teach helps in the industry, as does having a degree. You need to know how to manage and motivate people in academia. Coming back to academia, you start asking questions like, "Who cares about this project?" "Will it scale?" and so on.

Below are capsules of the discussion by the panel and the audience

Someone Whose Name I Forgot: People in academia are generally not interested in details, testing, and usability, which are needed to take something from research to a product. The industry in general is also not very interested in research work, reading papers, and so on.

Mark Day: More incentives are needed to get industry and academia to interact. Currently there are almost no such incentives.

Andrew Hume: I do technology transfer at AT&T. The problem is enticing researchers, because you go for a while without publishing papers. On the other hand, you can then write a different kind of paper, about the real aspects of systems. Academia should care more about results having to do with real details.

Noah Mendelsohn: Academic papers don't line up with industrial interests. If conferences did accept industry papers, will companies write them?

Andrew Hume: Yes. Motivating factors are satisfaction and recognition, perhaps because this is rare.

Noah Mendelsohn: There's also an opportunity cost to writing a paper, of losing developer time.

Charles Leiserson: Students going into industry don't understand company culture; they are used to the academic environment.

Margo Seltzer: There needs to be motivation for companies to write papers. Engineers want to write papers, but need to sell papers to managers, as a tool for marketing, for example.

Brian Bershad: Often companies don't want intellectual property published. Thus, commercial papers lack technical detail.

Charles Leiserson: Writing papers is usually as useful internally as getting it published. Papers help internal communication.

Roblis(?), Intel: At Intel Labs, writing papers is rewarded and expected. In the product groups, however, it is viewed as a net negative. It would be useful if conferences could accept/reject rough drafts to avoid wasted write-up efforts.

Anthropologists studied engineers and found that usually there are a few "leaders" in engineering groups that go to conferences, lead things to turn something into a product, etc. Maybe we don't need people transfer, we just need to market things to these "leaders"?

Dick Flower(?): There are groups without leading individuals. Having an advanced development group of some sort could be useful, though.

Brian Bershad: I think some companies have reasonable expectations of the research world, and some companies don't.

John(?): The HotChips conference, for example, only produces presentations and not papers. It's much easier to get a presentation, rather than a paper, from a lead chip designer.

Mark T (MS Research): At PARC, of the people who went to industry, none ever came back for long. Can you ever come back from the industry?

Brian Bershad: No, not possible to come back and be the same. Your focus changes to short-term goals.

Charles Leiserson: In my lab, lots of people, including staff, did it OK. Doing so colors your interest, though. You learn about things like barriers to adoption, etc.

Mark Day: Do you mean returning to applied research or to academia?

Mark T (MS Research): PARC returnees were successful in the industry and kept going back to form new startups. Focus on doing something with impact in the world.

Noah Mendelsohn: Having gone to industry before grad school gave me a great perspective on reality, judging the realism of projects, etc. It's very hard to do research part-time.

Bradley Chen (Appliant): What do you think about requiring faculty to have industrial experience?

Charles Leiserson: Depends on the quality of the experience. But yes, there are things from industry to be taught to university people. Management, leadership, motivation, educating about teamwork, working with each other.

Fred Douglis (IBM): Were there more core industrial papers back when USENIX took extended abstracts?

Chris Small: We seem to have a hangover after the dot-com boom. There was a huge flux of ideas from research to commerce.

Brian Bershad: The dot-com boom shows what happens when the barriers to adoption from research are removed. The result wasn't so great – too many worthless ideas, no industrial experience. Doing a startup is easier the second time around.

Someone from VMware: We were lucky to have good timing to submit our paper to OSDI – the submission deadline was a few months after an internal deadline, which gave us time to gather results. The community should be more receptive to papers about released or dead products; they are valuable.

Jun Nakajima (Intel): In this economy, R&D costs are being reduced and moved to China. For the cost of one engineer here you can get three to five engineers in China. How do you justify the three-to-five times cost?

Charles Leiserson: Education. Also location – most other companies are located here.

Erez Zadok (Stony Brook): Academia is not preparing students for life in industry. It's difficult to convince universities to create courses with practical aspects.

5th Symposium on Operating Systems Design and Implementation (OSDI '02)

TECHNICAL SESSIONS

DECENTRALIZED STORAGE SYSTEMS

Summarized by Himanshu Raj

FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment

Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, Roger P. Wattenhofer, Microsoft Research

The goal of this research was to make a scalable serverless distributed file system while maintaining security against malicious attacks in untrusted systems. Byzantine protocols are used to define untrusted infrastructure. The FARSITE solution is a virtual global file store of encrypted data that is replicated to facilitate availability. Storage is divided into two parts: file data and metadata. Metadata information is a hash-computed form of actual file data. The system is built around an infrastructure to store file data and metadata separately, and traditional Byzantine properties are applied only for machines storing metadata.

Since Byzantine operations are costly, they are not performed per file I/O. Instead, a Byzantine operation is defined valid for a period of lease. Various different types of leases are available to suit different consistency requirements. Batching is another concept used to reduce cost. The system is implemented as a service in user level and as a kernel mode driver that routes the actual file system calls to NTFS. According to the results reported, the system performs better than a central file system, though worse than bare-bones NTFS. The system is not designed to address efficient large-scale write sharing, database semantics, or disconnected operations. The project link is *http://research.microsoft.com/sn/Farsite/*.

Taming Aggressive Replication in the Pangaea Wide-Area File System

Yasushi Saito, Christos Karamanolis, Magnus Karlsson, Mallik Mahalingam, HP Labs

Pangaea is a scalable distributed file system targeted for the type of WAN infrastructure characteristic of multinational companies with overseas corporate offices and a need to share data. Design goals of Pangaea include hiding WAN link latencies, availability in a high-change environment, and network usage efficiency. The system assumes the presence of an available secure infrastructure, such as VPN. The system also provides eventual consistency, though manual open/close-style consistency could also be provided. The system employs pervasive replication to dynamically replicate each file/directory in the system independently. Benefits drawn from intensive replication are speed, availability, and network efficiency. The system is implemented in user space based on SFS API. The NFS client at the kernel level routes the I/O requests to the Pangaea service at the user level. The system uses graph-based replica management. A random graph is created for every file/directory in the system, and edges of this graph are used for update propagation among replicas and for replica discovery. Since the system does not have a central lock manager, it uses a technique called harbingers to compute a spanning tree so that duplicate transmissions can be avoided. This technique also helps reduce the propagation delay. The project link is *http://www.hpl.hp.com/research/ssh*.

Ivy: A Read/Write Peer-to-Peer File System

Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, Benjie Chen, MIT

The main goal of Ivy is to build a highly available file system out of inexpensive infrastructure that can scale to multiple writers on the same data. The system leverages DHT in the core, and provides weaker consistency guarantees for metadata for performance reasons. The main idea behind the system is to use a log per user and combine potentially multiple logs to serialize the updates made on a shared object. Serialization is based on a version-vectoring scheme rather than using a timestamp technique. The evaluation compares Ivy's performance with a local file system to see the load characteristics and runtime comparisons made with NFS over WAN. Results show that log operations tend to dominate the performance of the system over WAN and parallel fetching of log records can be used to hide latency. The system addresses sharing among only a small number of writers and hence does not address the scalability issues involved with a large number of writers sharing an object. Merging of logs is performed later, as in the Coda file system, and conflict resolution is addressed then. The way to provide effective read sharing in Ivy is to use multiple file systems. The project link is *http://pdos.lcs.mit.edu/ivy*.

ROBUSTNESS

Summarized by Ben Zhao

Defensive Programming: Using an Annotation Toolkit to Build DoS-Resistant Software

Xiaohu Qie, Ruoming Pang, Larry Peterson, Princeton University

Qie began by examining how typical DoS attacks work. One attacks a Web server, for example, by intentionally slowing down TCP, faking packet loss, and attempting to tie down as many TCP connections at the server end as possible. It is useful to classify resources as renewable (CPU, network bandwidth, disk bandwidth) and nonrenewable (processes, file descriptors, memory buffers). Renewable resources are vulnerable to "busy attacks," which try to request the resources faster than they can be allocated. The corresponding solution is protection via admission control. Nonrenewable resources are vulnerable to "claim-and-hold attacks," which attempt to request and hold on to them. The corresponding solution would be to recycle resources when they are exhausted, reclaiming them from certain applications. Combinations of the two types of resource attacks are harder to deal with. For example, when file descriptors (nonrenewable) are recycled to protect against claim-and-hold attacks, they become a renewable resource and therefore vulnerable to busy attacks.

The proposal is to utilize a toolkit containing "sensors" and "actuators" to protect both types of resources, with low programming burden. The toolkit is a combination of techniques from work in protection, static analysis, anomaly detection, and profiling. To protect renewable resources, the approach is to divide functionality into distinct services and balance resources among them, such that the impact of an attack on a single service is limited to that service. To protect nonrenewable resources, they need to be recycled when necessary. The algorithm to choose the resource instance to reclaim can be driven by a timer. The timer can be set on idleness or on the length of the service lifetime. The work proposes a user-defined progress metric (amount of data output or number of state transitions) that will reclaim resources from the "slowest" principal.

The toolkit is implemented as 11 C macros and library functions. The authors also modified gcc for auxiliary code generation at compile time. The evaluation contains case studies of a flash Web server. The Web server is partitioned into 46 services; 60 annotations were added to the code. Under a slash attack, the annotated server response time is 5.1 milliseconds, compared to a normal response time of 4.3 milliseconds, and is significantly lower than a non-annotated server under attack, which has a response time of 25 seconds. A possible limitation is that its effectiveness depends on service granularity. The project link is *http://www.cs.princeton.edu/nsg*.

Using Model Checking to Debug Device Firmware

Sanjeev Kumar, Kai Li, Princeton University

Device firmware is a piece of concurrent software that achieves high performance at the cost of software complexity. It contains subtle race conditions that make it difficult to debug using traditional debugging techniques. The problem is further compounded by the lack of debugging support on the devices. Model checking is a promising approach. It can systematically explore all possible scheduling orders and provide counter-examples of bugs found. The general technique is to extract models from programs either manually or via a compiler. The authors extracted models for the Spin model from programs written in the ESP language. ESP is a language for programmable devices that the compilers use to generate tests. In evaluation, the techniques are applied to VMMC (high performance communication design that bypasses the OS for data transfers). VMMC firmware was reimplemented using ESP. Seven models were found using abstract models, despite the global nature of some bugs (deadlock). These bugs would be hard to find without using a model. Where a full search of the state space is not possible, partial searches can minimize resource costs and still produce useful results.

CMC: A Pragmatic Approach to Model Checking Real Code

Madanlal Musuvathi, David Y.W. Park, Andy Chou, Dawson R. Engler, David L. Dill, Stanford University

Many system errors do not emerge unless some intricate sequence of events occurs. In practice, this means that most systems have errors that only trigger after days or weeks of execution. Model checking is an effective way to find such subtle errors. This work contributes the C model checker, which links to code, emulates a real system, captures the states of the system, and analyzes the results. CMC schedules threads to emulate nodes in the network, where scheduling granularity is on the order of entire even handlers. This means handlers are treated atomistically, and synchronization bugs can be missed. CMC tries to search the entire space, but it can checkpoint at decision points and resume later where different states can be generated. The work uses three optimizations to reduce the search space: hash compaction, downscaling, and state canonicalization. Hash compaction is the use of hashtables to store previously seen states so they are not examined again, and computing hashed signatures for each state to reduce space requirements. Downscaling is the use of a small number of nodes in order to reduce the state space. Complex interaction bugs can still be produced, but it might miss bugs only seen on large-scale interactions. State canonicalization is the simplification of similar states down to a single state, which is then evaluated. When applied to AODV routing protocol implementations, the CMC checker found 42 bugs (of which 34 are distinct, and one is a bug in the specification).

KERNELS

Summarized by Charles P. Wright

Practical, Transparent Operating System Support for Superpages

Juan Navarro, Rice University and Universidad Católica de Chile; Sitaram Iyer, Peter Druschel, and Alan Cox, Rice University

Translation lookaside buffer (TLB) coverage has decreased by a factor of 1000 in 15 years. In 1985 the TLB miss overhead was less than 5%; today it is over 30%. This is primarily due to increases in the size of working sets, yet TLB size has remained constant. Many architectures allow the creation of superpages. A superpage TLB is like a normal TLB, but a size field is added. Navarro presented a practical implementation of superpages for FreeBSD 4.3.

There are three major issues when implementing superpages: (1) superpages require allocation of contiguous, aligned memory; (2) a superpage can be created out of several normal pages (promoted) or broken into several pages (demoted); and (3) the need to prevent internal fragmentation. Each issue is dealt with in an opportunistic manner. For example, once an application touches the first page of a memory object it will quickly touch every page. Each superpage is created as long as possible and at the earliest point. To do this, reservation is employed, but it may be broken if the memory is needed (the oldest reservations are broken first). The same type of opportunistic algorithm is applied to promotion and demotion. To keep fragmentation low the page demon restores contiguity, and wired pages are clustered. On the SPEC CPU 2000 integer and floating point operations, a performance improvement of about 11% was observed. For a large matrix transposition, an improvement of over 600% was observed.

More information is available at *http://www.cs.rice.edu/~jnavarro/superpages/*.

Vertigo: Automatic Performance-Setting for Linux

Krisztián Flautner, ARM Limited; Trevor Mudge, University of Michigan

Flautner presented a software framework to do energy management by setting processor speed. The processor consumes 32% of the power budget on small devices (e.g., PDAs). Vertigo focuses on power management when the CPU is performing work, not when the CPU is idle. The underlying principle is to run just fast enough to meet deadlines, without using higher power consumptions. An increase in performance will create an exponential increase in energy usage—it is better to use a smaller amount of computing power for a longer period of time than to use a large amount of power over a short period.

Vertigo is a Linux kernel module that monitors system execution to determine how fast things need to go. There are five hooks in the kernel (e.g., task switching, some system calls, and swapping) that are used to determine activity. A policy stack combines multiple simple algorithms to determine the best performance level. Each algorithm can be specified for a specific performance situation. For example, an interactive performance algorithm may monitor X server events.

Vertigo was compared to the Crusoe LongRun on-chip power saving system. Using application-specific knowledge was very effective. For example, the Crusoe LongRun would cause spikes to full power when the GNOME clock ticked. When playing MPEG movies both Vertigo and LongRun do not drop any frames, but Vertigo used 52% of the peak performance level and LongRun used 80%. The conclusion is that the kernel has lots of valuable information that is lost on the chip.

Cooperative I/O: A Novel I/O Semantics for Energy-Aware Applications

Andreas Weissel, Bjorn Beutel, and Frank Bellosa, University of Erlangen

Traditional operating system power management assumes the timings of disk operations by user applications are unknown and cannot be influenced. Additionally, transitioning to a low-power mode will actually waste power if the transition was unnecessary. Cooperative I/O changes this assumption by introducing three new system calls: coop_read, coop_write, and coop_open. Along with the standard parameters for these calls, a timeout and an abortable flag are passed (e.g., a MPEG player may specify that I/O can be deferred until the frame actually needs to be decoded). This allows the operating system to schedule I/O intelligently.

There are three components to cooperative I/O: a modified IDE driver that shuts down the disk after the break-even point (the number of seconds required to reduce overall power consumption), VFS modifications, and ext2 modifications. The goal of these modified components is to cluster I/O operations into batches, thus leaving the drive idle for the longest period of time possible. For the Amp MP3 player, 150 lines of code were modified (the bit rate was used to determine timeouts). While this modified Amp was running, an unmodified mail client using write was used. Using coop_read, a power consumption was reduced to 210 joules from 373 joules. This is a better energy savings than an "Oracle" policy and one which always makes the right power decision based upon a previous trace, without modifying the timing of the I/O.

PHYSICAL INTERFACE

Summarized by Charles P. Wright

TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks

Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, University of California, Berkeley; Wei Hong, Intel Research

Sensor networks are a collection of small, inexpensive battery-run devices with sensors and RF interfaces. Programming a sensor network is a difficult task: It took two weeks for two experienced students to program a vehicle-tracking sensor network. TAG eliminates the need to program sensor networks by using an SQL-like declarative language—using TAG, the same vehicle position network was programmed in two minutes. Sensor networks are installed under harsh conditions (e.g., in habitat- or earthquake-monitoring applications). The primary metric used for sensor networks is power consumption. Berkeley "Mica Motes" run for only two to three days when using full power but can last up to six months at a 2% duty cycle. Communication dominates the power consumption cost, so they use bytes sent as a metric.

To reduce the communications overhead, TAG allows in-network processing of aggregate queries (count, max, average, etc.). Madden asserts that most common data-analysis operations are aggregate operations. For example, the average temperature over all the sensors (or in a given sector) is a more interesting indicator than the temperature at each individual node.

There are several methods that can be used to decrease communication. The first method is to incrementally compute values using partial state records (PSRs). For example, an average can be transmitted to a node's parent as a sum and a count, and the parent's values can be inserted into this PSR. Additionally, snooping or guesses can improve performance. If the desired aggregate is the max, a node does not need to communicate its own value if it hears a value larger than its own. If the root knows the max value is at least 50, then it can reduce communication by communicating this value to other nodes.

More information can be obtained at *http://telegraph.cs.berkeley.edu/tinydb/*.

Fine-Grained Network Time Synchronization Using Reference Broadcasts

Jeremy Elson, Lewis Girod, Deborah Estrin, UCLA

To present a consistent view of information, sensor networks need to have a consistent view of time. This problem has already been solved on the Internet (e.g., NTP), but sensor networks do not have the infrastructure available to Internet hosts. Sensor applications also have stronger time synchronization requirements than the Internet (tracking phenomena may require microsecond-level synchronization).

Elson presented reference broadcast synchronization (RBS). Traditional synchronization methods have lots of nondeterministic delay when sending packets (e.g., backoff timers or link-level retransmission). Receiving a packet that a host sent has much less variation than the time it takes to actually send a packet

(1 bit width for receive vs. 1000 for send). Therefore, two hosts can make note of the time they received a packet sent by a third host. The two receivers now know the difference between their clocks. Clock skew perturbs this observation, however, so a best-fit line is used to determine the difference.

RBS synchronized the clock on a Compaq iPaq to precisions of 6 microseconds, whereas NTP is only able to obtain a precision of 53 microseconds. The clock resolution on the Linux platform is only 1 microsecond; Elson believes that a more accurate clock would yield better results. The performance under a 6 Mbps load shows even better results: RBS degrades to 8 microseconds, but NTP degrades to 1542 microseconds.

RBS effectively removes sender nondeterminism from network time synchronization. This facilitates a wide range of applications, including acoustic ranging and collaborative signal detection.

SUPPORTING TIME-SENSITIVE APPLICATIONS ON A COMMODITY OS

Ashvin Goel, Luca Abeni, Charles Krasic, Jim Snow, Jonathan Walpole, Oregon Graduate Institute

Fast processors enable interactive real-time applications in software: for example, software radio, software modems, voice over IP, video conferencing, and accurate network traffic generators. However, these applications need millisecond to microsecond timing guarantees. It has long been accepted that to provide such timing guarantees a special real-time OS is required and that general purpose OSes need a complete redesign. Real-time operating systems have many disadvantages (e.g., nonstandard interfaces and small user communities). Goel presented time-sensitive Linux (TSL). TSL aims to provide real-time performance on commodity general-purpose operating systems using an evolutionary approach.

The requirements for TSL were fine-grained timers, a responsive kernel, and an accurate implementation of a good scheduler. There are two types of kernel timers: fine-grained (soft) or one-shot timers (firm). There are two overheads when evaluating timers: reprogramming and interrupts. Reprogramming the timer turns out to be inexpensive, but the interrupts are expensive. However, soft timers have a potentially unbounded latency. TSL uses firm timers. Firm timers insert checks into kernel paths (e.g., system call entry/exit to check the timer) but also use one-shot timers that are configured to overshoot the required delay. This provides some guarantee while, hopefully, reducing the number of interrupts.

Linux has already broken down the big kernel lock, but some locks are still large. TSL uses voluntary lock yielding to increase kernel responsiveness. Finally, TSL implements a proportional share scheduler that provides a constant-speed virtual machine. Even heavy-load software modems (which require 4–16 millisecond guarantees) are supported. TSL implements sub-millisecond-timing guarantees on a general-purpose operating system. TSL imposes 1.5% overhead, which is low for firm timers. For the additional kernel preemption points, an overhead of 0.5% was introduced.

PANEL

Summarized by Steven Czerwinski

SELF-ORGANIZING NETWORKS FROM SENSOR NETS TO P2P: PANACEA OR PIPE-DREAM?

Co-Moderators: Peter Druschel, Rice University; David Culler, University of California, Berkeley/Intel; Panelists: Hari Balakrishnan, MIT; Yaneer Bar-Yam, NECSI; John D. Kubiatowicz, University of California, Berkeley

Are self-organized networks superior to traditionally engineered solutions and are they necessary to solve today's problems? In Culler's opinion, self-organized networks are necessary but require engineering in order to build systems with the desired predictable global behaviors. They are necessary in sensor networks because of the near impossibility of administering and configuring millions of sensor nodes; they must be able to self-organize. However, as he showed with several different self-organizing methods to compute spanning trees, designing the local rules that lead to correct global behavior is difficult. This is where engineering needs to be applied.

Balakrishnan also advocated self-organized networks because they can eliminate human misconfiguration from distributed systems and allow such systems to adapt to errors and change. He argued that distributed systems are all about enabling autonomy at subsystems, but with this autonomy come problems with misconfiguration. Using traces, he showed how a significant portion of invalid DNS queries were caused by human misconfiguration.

Kubiatowicz used a thermodynamics analogy to argue the importance of self-organizing networks. Large systems can exhibit stability through statistics if they possess replicated components that interact and adjust to one another. Energy could be injected into the system through both passive and active correction mechanisms. He labeled such systems as "thermospective." With Moore's Law enabling redundancy and with the need to eliminate human configuration, he saw these systems as being the future.

Druschel presented a spectrum of current distributed systems, with decentralized approaches on one end and self-organizing ones on the other. He argued that natural (biological) systems are the only truly self-organizing systems, with sensor networks being fairly close. Systems requiring ACID semantics have difficulties making it onto the self-organized end. He also noted that the systems we engineer are robust to both mundane failures and malicious attacks, while self-organizing ones are only robust to mundane failures. They would require (at the least) a trusted certificate authority to be robust to attacks.

Bar-Yam used analogies from biology to show that we already have the conceptual tools to demystify self-organizing networks. It may be hard to understand the progression of a mouse embryo from a macroscopic perspective, but that's because we don't understand the local rules or patterns of behavior of the smaller components. He showed how different types of patterns of behavior (such as local majority, two-dimensional condensation, and local activation/long-range inhibition) can lead to interesting phenomena, such as the stripes on a zebra's back.

Audience members pointed out the difficulties of creating such a system from an economic and business standpoint (who pays for all of this?) along with privacy concerns (do you really want your data going anywhere and everywhere?). Some also cautioned against the misuse of biology and other non-computer science metaphors, which can encourage similarities being drawn where none exist.

VIRTUAL MACHINES

Summarized by Praveen Yalagandula

Memory Resource Management in VMware ESX Server

Carl A. Waldspurger, VMware

This won the Best Paper award.

VMware ESX server is a thin kernel to multiplex hardware resources among virtual machines. The three main issues that arise in memory resource management are fairness, performance isolation among virtual machines, and efficient utilization of the available machine memory.

To efficiently reclaim memory from a virtual machine, Waldspurger proposes the ballooning technique where a driver inside the virtual machine allocates some pages, forcing the guest OS to evict pages not in use or to swap some pages. Experimental results show that there is only a small overhead of 1.4% to 4.4% in using this technique.

Efficient use of available machine memory is provided through memory sharing, where a single page on the machine is shared by multiple VMs (using copy-on-write semantics). A background process computes hashes of pages to determine the duplicate pages. For "best case" workloads, in which multiple Linux VMs are run, about 60% memory savings are observed. For real workloads, the savings ranged from 7% to 32%.

For a memory allocation scheme that provides fairness among virtual machines while being efficient, the author proposes the concept of "idle memory tax," where the idle pages are charged more than active pages. This new mechanism resulted in a 30% throughput increase for the workload considered in experiments.

Scale and Performance in the Denali Isolation Kernel

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

The goal of this work is to enable the execution of untrusted code while providing isolation so that the untrusted code does not interfere with any other process on the system. The Denali "isolation kernel" isolates untrusted software services in separate protection domains. The approach is to use virtual machines to provide isolation with strategic modifications for scalability, simplicity, and performance.

Denali's virtual machine architecture achieves scalability and performance at the cost of giving up backwards compatibility. It omits rarely used features like BIOS, protection rings, etc.; revises interrupts and MMU; and simplifies hardware I/O instructions. The resulting core kernel is an order of magnitude smaller than the bare-bones Linux 2.4.16 kernel.

For scalability, Denali employs the following techniques: (1) batched, asynchronous interrupts – instead of invoking a VM when interrupt arrives, interrupts are batched together and applied when the corresponding VM is scheduled, thus reducing the overhead of context switches; and (2) idle-with-timeout instruction – this instruction allows VM to specify how long it yields, thus leading to better scheduling. The first technique provided a 30% improvement in performance in experiments, and the second scheme yielded a 100% throughput improvement.

ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay

George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen, University of Michigan

The aim of this work is to provide a way for post-mortem analysis of intrusions. Typical system logs are subverted by the intruder. The "CoVirt" project aims at enhancing security by running the target OS and all target services inside a Virtual Machine (VM) and then adding security services in the VM or host platform. ReVirt aims at checkpointing and logging a VM's execution trace so that it can be replayed later. The virtual machine used is UMLinux, a Linux kernel that can be run on any other Linux machine.

To enable complete replay, checkpointing is done that covers the memory, CPU, and disk states, and logging is done that covers all keyboard, network events, and interrupts, along with the data corresponding to these events. Replaying the interrupts is a hard problem, and the authors use tuple, as in the Hypervisor project, to uniquely identify the place in execution where interrupts should happen.

The virtualization overhead ranged from 1% to 58% for different workloads. The logging overhead on runtime is about 8%, and the log grew at a rate of 1.4GB/day in the worst case workload and at 0.04GB/day in the best case.

CLUSTER RESOURCE MANAGEMENT

Summarized by Praveen Yalagandula

Integrated Resource Management for Cluster-based Internet Services

Kai Shen, University of Rochester; Hong Tang,University of California, Santa Barbara; Tao Yang, University of California, Santa Barbara and Ask Jeeves; Lingkun Chu, University of California, Santa Barbara

The challenges involved in hosting large-scale resource-intensive Internet services on a server cluster are: (1) scalability and robustness, (2) timely response, (3) efficient resource utilization, (4) adaptive resource management, and (5) differentiated services. The goal of the Neptune project is to provide programming and run-time environment support for effective management of services through partitioning, replication, and aggregation. Instead of using monolithic metrics such as throughput, mean response time, etc., the authors define "quality-aware service yield" with respect to a request as denoting the amount of economic benefit resulting from servicing this request in a timely fashion, and then try to maximize the aggregate service yield over all requests.

Service differentiation is done based on service classes, where service accesses of a particular service class obtain the same level of service support. A service class can be a set of client identities, service types, or data partitions. In Neptune, two-level request distribution and scheduling is done: gateways do random polling of the servers and try to achieve load balancing, and service differentiation is done at the servers. This two-level architecture provides scalability and robustness at the cost of less isolation and fairness. Within a server, a request scheduler schedules requests from the queues belonging to different classes to several worker threads such that the aggregate yield is maximized. The offline optimal scheduling problem is NP-complete, and, hence, the authors use heuristics such as Earliest Deadline First (EDF), Yield-Inflated Deadline (YID), Greedy, and Adaptive techniques. The experimental results show that Adaptive outperforms all other heuristics on a 16-node cluster.

Resource Overbooking and Application Profiling in Shared Hosting Platforms

Bhuvan Urgaonkar, Prashant Shenoy, University of Massachusetts; Timothy Roscoe, Intel

The goal of this work is to maximize the number of hosted applications on a server cluster while providing resource guarantees to the applications. Taking a worst-case load and assigning those amounts of resources is not efficient, since the average load of an application is typically an order of magnitude less than the worst case. So the authors propose to use the scheme of overbooking resources and show that this scheme is feasible and maximizes the revenue generated by the available resources.

The authors define "capsules" as the components of an application that runs on a node. To determine the resource requirements of a capsule, the authors perform "application profiling" using Linux Trace Toolkit (for CPU and memory requirements) with well known traces. From typical application profiles, the authors conclude that these capsules exhibit different degrees of burstiness and use "Token Buckets" to represent the resource requirements. A Token Bucket of a capsule with two parameters *s* and *p* states that the resource usage of that capsule over any time period *t* has to be $<= s*t + p$. Each capsule specifies an overbooking tolerance parameter, *O*, to denote the probability with which the resource requirements of that capsule can be violated. Once capsules' resource requirements are estimated, these are mapped to nodes using a simple algorithm that uses a greedy technique. A capsule can be mapped to a node only if the resource requirements of the capsule can be satisfied by the node. The experimental results show that there is a 100% improvement with just 1% overbooking.

An Integrated Experimental Environment for Distributed Systems and Networks

Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar, University of Utah

Typically, network experiments are done through simulation, emulation, or on live networks. While simulation is repeatable but not accurate, live network experimentation is realistic but not repeatable. The emulation method of experimentation is a hybrid approach that creates a synthetic network environment but requires tedious manual configuration. Netbed complements existing experimental environments by spanning simulation, emulation, and live experimentation, integrating them into a common framework. The integration allows ease of use while being realistic. About 2176 experiments were done on the Netbed within the last 12 months by about 365 users.

The Netbed uses a virtual machine approach for network experimentation. Configuration time is improved through automation by two orders of magnitude. Network nodes are emulated using virtual machines on a cluster of nodes. Links including WAN links are emulated using VLANs and tunnels. The network topology to be emulated can be specified either using a ns-type Tcl-based specification or in a Java-based GUI. A global resource allocator scheme assigns local cluster resources to different components of the network topology requested.

Configuring a six-node dumbbell network took just 3 minutes on Netbed, in comparison to a 3.5-hour effort by a student with significant Linux system administration experience.

For more information, see *http://www.netbed.org*.

PEER-TO-PEER INFRASTRUCTURE

Summarized by Scott Banachowski

Scalability and Accuracy in a Large-Scale Network Emulator

Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, David Becker, Duke University

Yocum discussed a network traffic emulator designed to provide realistic scenarios for complex systems such as the Internet. Using the emulator, called ModelNet, has advantages over simulation because it allows execution of real code while still providing control over network conditions not possible with live deployment. The goals when developing ModelNet included support for 10K nodes with a 10Gbps bisection bandwidth, and realistic emulation of network failures and cross-traffic.

The emulator organizes networks into two types of nodes: (1) edge nodes that run the code being tested and connect through (2) core nodes that run ModelNet emulation code. A technique called "distillation" is the key for providing the scalability necessary for handling large numbers of nodes. Distillation transforms the topology of core nodes, which represent the Internet, into a smaller subset of nodes that preserve only interesting links, including the first and last hops of the edge nodes. In this approach, instead of injecting packets that incur processing overhead for an emulator, cross-traffic is simulated by changing the characteristics of the connections through the core nodes.

The ModelNet emulator was verified by reproducing experiments from a previously published study of the CFS storage system layered on the Chord distributed hashtable. Running Chord/CFS on the edge nodes and substituting ModelNet for the network, the throughput of data transfers closely matched the previously published results. Yocum concluded with the assertion that ModelNet is effective for studying how your code behaves in a large-scale network running on its native OS. Questions from audience members revealed that it was not known yet exactly how far ModelNet scales, and that it does require a lot of storage.

More information is available at *http://issg.cs.duke.edu/modelnet.html*.

Pastiche: Making Backup Cheap and Easy

Landon P. Cox, Christopher D. Murray, Brian B. Noble, University of Michigan

Users rarely, if ever, make backups of their personal systems, because it is expensive and time-consuming. Capitalizing on the trend that many disks are often less than half-full, Pastiche is a system for peer-to-peer backups of files on others' computers. Recognizing that many of the binaries on a disk are identical to the binaries of other users, much of the cost of transferring data is eliminated. The goal of Pastiche is efficient, cost-effective backup, while preserving individual privacy.

As its name implies, Pastiche is assembled from already existing technologies. Pastiche uses content-based indexing of data, the same techniques employed by LBFS. Data is fingerprinted and divided into chunks, and a hash function uniquely identifies each chunk. Using only a subset of fingerprints from a disk – for example, a fingerprint from a Windows distribution – Pastiche can identify redundant copies of the data on other machines. To locate machines for backing up data, or "backup buddies," Pastiche uses two overlay networks determined by Pastry, a peer-to-peer routing infrastructure. A mechanism called "lighthouse sweep" was added to Pastry to ensure a geographically diverse set of nodes.

When participating in Pastiche, your system may contain information that also backs up your peers' systems, so the file system must ensure that this data is not deleted or modified. The Chunkstore file system views all data as chunks and assembles files for users in objects called "container files." When data from a container is modified, it is written to a new chunk, preserving the older versions of the data. The performance of backup and restore operations is comparable to VFS copies.

The talk generated enough controversy that there were long lines at the questioning microphones, mostly people interested in more in-depth comparisons with other backup methods.

Secure Routing for Structured Peer-to-Peer Overlay Networks

Miguel Castro, Microsoft Research; Peter Druschel, Rice University; Ayalvadi Ganesh, Antony Rowstrom, Microsoft Research; Dan S. Wallach, Rice University

While peer-to-peer overlay networks are scalable, self-organizing, and robust with respect to node failure, they are susceptible to malicious participants. The talk presented several attacks on these overlays followed by a discussion of defenses.

Castro began with an overview of the Pastry routing overlay and then described several attacks on this technique. In one type of attack, a node can choose its node ID so that, instead of being random, it is positioned to control another node's network access or prevent availability of objects. A defense against this attack would be to certify node IDs using keys from a trusted source. To prevent users from obtaining a large number of node IDs, certificates, it was suggested, might require purchasing. Other attacks on overlays affect routing: for example, supplying peers with fake proximity information or bad routing table information to increase the probability that messages travel through a malicious node. A defense for attacks on routing is to maintain a fallback table, with constrained and more verifiable routing for use when the performance-based routing table fails. Finally, a malicious node may drop or misroute messages. A solution is to incorporate a routing test, and if it fails, rely on a redundant route.

Using these security techniques, peer-to-peer protocols may still work even when up to a quarter of the nodes of an overlay network are malicious, and they provide efficiency when the actual number of compromised nodes is small. In the question period, one audience member quipped that the idea of charging for certificates was the work of the presenter's employer and suggested that the alternative of using a real-world authentication based on a user's identity is more viable.

WORK-IN-PROGRESS REPORTS

Summarized by Scott Banachowski

DISCOVERING BOTTLENECKS IN DISTRIBUTED SYSTEMS

Athicha Muthitacharoen, MIT; Jeffrey C. Mogul, Janet L. Wiener, HP Labs

Contact: Athicha Muthitacharoen, *athicha@amsterdam.lcs.mit.edu*

In large, distributed systems it is not always possible to investigate causes of performance bottlenecks created by internal, proprietary components, because discovering problems often requires instrumenting these components to measure statistics. MIT is developing a tool to identify critical paths using a passive trace of messages. Using the relationships between messages, the tool automatically infers the source of bottlenecks.

WITNESS: LEADER ELECTION WITHOUT MAJORITY

Haifeng Yu and Amin Vahdat, Duke University

Contact: Haifeng Yu, *yhf@cs.duke.edu*

The title must have been inspired by the last presidential election. Many distributed algorithms require a node be elected as leader, but under some kinds of failures it is impossible to guarantee that the elected leader is unique. The new election algorithm provides probabilistic guarantees of a unique leader, and is based on choosing a random set of witnesses to participate in the protocol.

CONFIDENTIAL BYZANTINE FAULT-TOLERANCE

Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, Mike Dahlin, University of Texas, Austin

Contact: Arun Venkataramani, *arun@cs.utexas.edu*

As replication systems add more servers and heterogeneity, they become increasingly vulnerable to attack, so providing confidentiality for replicated data is a difficult problem. This system increases the intrusion-tolerance of a set of replication servers when a number of the servers fail.

INCREASING FILE SYSTEM BURSTINESS FOR ENERGY EFFICIENCY

Athanasios E. Papathanasiou, Michael L. Scott, University of Rochester

Contact: Athanasios Papathanasiou, *papathan@cs.rochester.edu*

This report describes a method to create longer idle times in disk traffic so that these idle periods may be exploited for power saving. The key is to increase the burstiness of access using aggressive prefetching combined with new disk-scheduling algorithms. Trace experiments show the energy reduction using this technique during an MP3 playback reached 55%.

FAB: FEDERATED ARRAY OF BRICKS

Yasushi Saito, Svend Frolund, Arif Merchant, Susan Spence, Alastair Veitch, HP Labs

Contact: Yasushi Saito, *ysaito@hpl.hp.com*

The talk described a logical disk system that uses low-cost commodity CPU and disks and is intended to replace high-end disk arrays. The decentralized system software, based on Petal, achieves high-performance and fail-over ability by replicating disk blocks throughout the cluster.

NCRYPTFS: A SECURE AND CONVENIENT CRYPTOGRAPHIC FILE SYSTEM

Charles P. Wright, Michael C. Martino, and Erez Zadok, Stony Brook University

Contact: Charles P Wright, *cwright@ic.sunysb.edu*

NCryptfs is a stackable file system based on CryptFS from FiST. The low-level file system is transparent to applications. An attach maps an accessed directory to its associated encrypted directory (which stores the actual data in cipher form). Each attach keeps its own data and authorizations private, and on-exit callbacks purge the clear-text data from the kernel.

SUPPORTING MASSIVELY MULTIPLAYER GAMES WITH PEER-TO-PEER SYSTEMS

Wei Xu and Honghui Lu, University of Pennsylvania

Contact: Honghui Lu, *hhl@cis.upenn.edu*

A massively multiplayer game supports up to 200,000 players. Traditionally, games use a client-server architecture, but Wei Xu proposes using peer-to-peer protocols. The talk describes a mapping of players to subsets of multicast groups. By trading consistency for performance, only "nearby" players need to synchronize their environments using P2P multicast groups. A prototype game was developed using Scribe.

KELIPS: A FAT BUT FAST DHT

Indranil Gupta, Prakash Linga, Dr. Kenneth Birman, Dr. Al Demers, Dr. Robbert Van Renesse, Cornell University

Contact: Indranil Gupta, *gupta@cs.cornell.edu*

Kelips is a peer-to-peer probabilistic protocol for group discovery, in which the lookup cost of a file is reduced by enabling the address of any file to be discovered within a single hop. This is achieved by increasing the size of file index tables on each peer and using background communication, or "gossiping," between nodes to keep state updated.

Improvised Network: Autonomously Reconfigurable Mobile Network

Nobuhiko Nishio, Keio University, Japan

Contact: Nobuhiko Nishio, *vino@sfc.keio.ac.jp*

New applications are emerging that use a combination of wireless networks and distributed sensor nodes, as in cellular phones. In such an ad hoc network, both sensors and sink nodes may be mobile, so the research is developing ways to adapt to the changing environment without hurting performance.

Probabilistic Energy Saving in Sensor Networks

Santashil PalChaudhuri and David B. Johnson, Rice University

Contact: Santashil PalChaudhuri, *santa@cs.rice.edu*

On mobile devices, idle and receive periods use about the same amount of energy, so if idle periods may be replaced with inactivity, the device stands to save a lot of energy. According to the "birthday paradox," a relatively small number of people can ensure a high probability that two of them share the same birthday. Applying this principle to communication, only a small number of nodes is needed to ensure that a sender and receiver are active simultaneously. Using a probabilistic-based protocol, the device pre-chooses its waking and sleeping periods, introducing some increase in communication latency but drastically reducing power consumption.

Solar: Supporting Context-Aware Mobile Applications

Guanling Chen and David Kotz, Dartmouth University

Contact: Guanling Chen, *glchen@cs.dartmouth.edu*

The goal of this research is to provide flexible and scalable pervasive computing. Solar is an infrastructure for context computation. An example is a mobile device that subscribes to a set of interesting events; in Solar, by moving the processing of these events to the infrastructure (called "planets"), applications that subscribe to the events remain lightweight. Sharing the computation among several applications reduces both development and network costs.

The exNode Distribution Network

Jeremy Millar, University of Tennessee

Contact: Jeremy Millar, *millar@cs.utk.edu*

exNode is a content distribution network. It is developed to provide access to time-limited data, such as the release of a software product, and is currently used by RedHat. The exNode architecture is effective at distributing load by implementing a highly distributed wide-area RAID system.

Scalable Constrained Routing in Overlay Networks

Xiaohui Gu and Klara Nahrstedt, University of Illinois, Urbana-Champaign

Contact: Xiaohui Gu, *xgu@cs.uiuc.edu*

This system is a step toward value-added service overlays. In overlay networks, such as those used by peer-to-peer applications, it is desirable to satisfy some end-to-end constraints – for example, establishing a level of quality of service between endpoints. Qualay is a proposed overlay network designed to provide QoS constraints over paths. In the setup phase, service paths are chosen by probing nodes, and in the runtime phase, faults are detected and paths rerouted to maintain QoS.

Reverse Firewalls in Denali

Marianne Shaw and Steve Gribble, University of Washington

Contact: Marianne Shaw, *mar@cs.washington.edu*

Shaw presented a way to introduce policies and mechanisms to protect the Internet from bad services. The system allows untrusted code to run in the network infrastructure on a virtual machine, with a reverse firewall that prevents the Internet from malicious traffic generated by the VM. The flexible framework allows policies to be added on the fly, and in the example provided in the talk, Shaw focused on a "don't speak unless spoken to" policy for containment of client-server code.

Improving Application Performance Through System Call Composition

Amit Purohit, Joseph Spadavecchia, Charles Wright, Erez Zadok, Stony Brook University

Contact: Amit Purohit, *purohit@cs.sunysb.edu*

A problem with application performance is overhead incurred by system calls that move data across the kernel boundary. This system provides a solution that removes user-level bottlenecks by moving user code into the kernel. Using a tool called Cosy, combined with the gcc compiler, designated code is compiled into special code segments that can be loaded into the kernel at runtime. Static and dynamic checks ensure that kernel security is not violated, and adding preemption to the kernel protects against user segments monopolizing the CPU.

Performance of Mach-Kernel

Igor Shmukler, OS Research

Contact: Igor Shmukler, *shmukler@mail.ru*

Shmukler spoke about enhancements to the Mach kernel aimed at increasing its attractiveness to the user community. Although Mach introduced many good ideas, it didn't really catch on because it was never fine-tuned for the common-case performance. Shmukler tried to clear Mach's bad name by discussing proposed improvements, including changing the memory management subsystem, optimizing the RPC implementation, adding new synchronization primitives, and stomping on a slew of bugs.

Elastic Quotas

Ozgur Can Leonard, Jason Nieh, Erez Zadok, Jeffrey Osborn, Ariye Shater, Charles P. Wright, Kiran-Kumar Muniswamy-Reddy, Stony Brook University

Contact: Jeffrey R. Osborn, *jrosborn@ic.sunysb.edu*

"Elastic quotas" for disks are aimed at shared file servers, such as those used by university students, where each user receives a quota of space. By implementing elastic quotas, extra space may be allocated to users for their temporary use, but this space may later be reclaimed. The elastic quota service sets both global and user-assigned policies for how the space occupied by files designated as elastic will be reclaimed, using information such as size or creation time. The next step in their research is to determine if users will embrace such a system.

A Mail Service on OceanStore

Steven Czerwinski, Anthony Joseph, John Kubiatowicz, University of California, Berkeley

Contact: Steven Czerwinski, *czerwin@eecs.berkeley.edu*

The Mail Service uses the OceanStore file system to provide low-latency access to email, independent of a user's location. Goals of the system include data durability and relaxed consistency by allowing application-specific conflict resolution. Following this session, members of the project gave a demo of OceanStore.

NETWORK BEHAVIOR

Summarized by Kenneth Yocum

An Analysis of Internet Content Delivery Systems

Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, Henry M. Levy, University of Washington

There's a lot more than just Web content being served across the Internet. Now we have CDNs and peer-to-peer systems serving up audio, video clips, and movies. The authors studied HTTP Web traffic, Akamai CDN, and Kazaa and Gnutella nets. The basic result of the authors' trace, conducted at the University of Washington, is that peer-to-peer traffic constitutes a large fraction of the bytes, and it's very different from the Web. For example, it may be possible to cache 80–90% of the outbound traffic and 60% of the inbound traffic. But it takes a long time to warm up the cache (about a month). In both directions P2P objects are three orders of magnitude larger than Web objects. A small number of objects account for most of the bytes in P2P systems.

TCP Nice: A Mechanism for Background Transfers

Arun Venkataramani, Ravi Kokku, Mike Dahlin, University of Texas, Austin

TCP NICE, a building block for background transfers, finds and uses spare bandwidth in the Internet to improve availability, reliability, latency, and consistency. As a "new" variant on TCP congestion control, TCP NICE is similar to TCP VEGAS monitor RTT but provides three changes: a more sensitive congestion detector, multiplicative reduction in response to increasing RTT, and the possibility of having a congestion window less than one. With NICE you can bound the interference caused by background flows. One use is prefetching. The authors found that NICE could improve performance by a factor of three, where using old-style TCP hurt performance by a factor of six.

The Effectiveness of Request Redirection on CDN Robustness

Limin Wang, Vivek Pai, Larry Peterson, Princeton University

We now use replication across geographic distance to deliver content. Client requests are delivered to the "best" candidate based on server load, server closeness, and cache. This work describes current schemes and introduces a new one to balance locality, load, and nearness (proximity). This new scheme was shown through simulation to improve system capacity by 60–90% while maintaining low request latencies for clients. One dynamic algorithm, Fine Dynamic Replication (FDR), is especially promising. It keeps fine-grained information on URL popularity to adjust the number of replicas. They're trying to deploy it on PlanetLab.

MIGRATION

Summarized by Richard S. Cox

The Design and Implementation of Zap: A System for Migrating Computing Environments

Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh, Columbia University

Zap supports the transparent migration of unmodified applications. The migration of network applications is supported without loss of connectivity. Zap-migrated processes leave no residual state behind on the previous system. Implementing Zap involves minimal changes to a commodity operating system and requires low overhead.

Three problems must be solved to migrate processes: resource consistency, resource conflicts, and resource dependency. Zap's solution to all three is the process domain (pod). A pod is a private virtual space that may contain a single process, a process group, or a whole user session. As a private space, processes in a pod cannot interact with processes outside a pod. Pods are migrated as a unit. Zap contains pods by introducing a thin layer in the Linux kernel, virtualizing process IDs, IPC, memory, the file system, network, and devices. The overhead of this approach is minimal, and the pod images are small.

More information can be found at *http://www.ncl.cs.columbia.edu/research/migrate*.

Optimizing the Migration of Virtual Computers

Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, Mendel Rosenblum, Stanford University

By virtualizing the x86 architecture, the VMware GSX server enables an entire virtual machine's (VM) hardware state to be easily suspended and captured. Once saved, the state can be sent to another machine and resumed. However, capturing the entire state generates machine images, or capsules, that are gigabytes in size. This work applies several optimizations to reduce the capsules to a size that can be transferred over a DSL link in under 20 minutes, enabling applications such as user mobility and software updates. The two largest components of a capsule are the disk and memory images.

Using standard copy-on-write techniques, VMware can track the changes to a disk image and transfer only the differences if the target machine already has an old version of the disk image. By hashing each disk block, and searching for a block with matching hash value on the target system, the server can avoid transferring pages whose contents already exist on the target system. Much of a VM's memory may not be in active use; thus, if VMware could request that the guest OS de-allocate inactive pages, the size of the memory image could be greatly reduced. This is the idea behind ballooning, which utilizes a driver added to the guest OS to reclaim low-priority pages prior to suspending the VM. Finally, by demand-paging the disk images, the time to resume the VM on the target can be reduced. Demand-paging takes advantage of the disk-latency tolerance already built into modern OSes. Several macro-benchmarks show that the combination of these techniques is effective in reducing the total data transferred to migrate a capsule as well as the time-to-start.

Luna: A Flexible Java Protection System

Chris Hawblitzel, Dartmouth College; Thorsten von Eicken, Expertcity

Extensible applications require protection schemes that can isolate extensions while permitting lightweight communication. Java uses language-based approaches to enforce domain separation, enabling cheap communication because of the single address space. However, systems with Java extensions lack clear domain boundaries; all code and objects are stuck together. The resources used by an extension cannot be reclaimed if the extension is terminated, because they may be referenced by other parts of the system.

By introducing a task abstraction, extensions in a Java system can be strongly isolated. Tasks contain all the objects, threads, and code for an extension. All cross-task communication is explicit. In Luna, regular (local) pointers are not allowed to reference objects in other tasks. Remote pointers, a new type of reference that is allowed to point to objects in other tasks, are Luna's mechanism supporting intertask communication. Remote pointers may be revoked at any time; if a revoked remote pointer is used, an exception is raised. This allows an entire extension to be removed from the system cleanly, without dangling references in other tasks. Remote pointers are implemented with a two-word structure. The first word is just the memory address of the object. The second word is a pointer to the permit, which contains a revocation flag and is checked before each use. As an optimization that removes most checks in common cases, Luna can generate loop code that does not contain any checks. On revocation, threads using the object are suspended, and a breakpoint is placed where the check would have been. If and when the breakpoint is reached, an exception is raised, simulating the effect of the check. Micro-benchmarks, as well as an implementation of an extensible Squid Webcache, confirm that Luna's isolation imposes low overhead.