conference reports

THANKS TO OUR SUMMARIZERS

OSDI ’08: 8th USENIX Symposium on Operating Systems Design and Implementation

San Diego, CA
December 8–10, 2008

The Jay Lepreau Award for Best Paper was instituted at OSDI ’08 (see p. 87). Three awards were presented.

CLOUD COMPUTING

Summarized by Dutch Meyer (dmeyer@cs.ubc.ca)

- ***DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language***
  Yuan Yu, Michael Isard, Dennis Fetterly, and Mihai Budiu, Microsoft Research Silicon Valley; Úlfar Erlingsson, Reykjavík University, Iceland, and Microsoft Research Silicon Valley; Pradeep Kumar Gunda and Jon Currey, Microsoft Research Silicon Valley

Winner of Jay Lepreau Award for one Best Paper

Yuan Yu explained that DryadLINQ comprises Microsoft’s cluster computing framework (Dryad) and the .NET Language INtegrated Query system (LINQ). Together, these tools allow programmers to write in their familiar .NET languages, with integrated SQL queries. In the compilation process, DryadLINQ can produce an executable appropriate for a single core, a multi-core, or a cluster of machines. The tool chain is incorporated into Visual Studio as well, allowing programmers to work in a familiar IDE. The result is an environment for concurrent programming that transparently works in the much more difficult field of distributed programming. In summary, Yuan described the work as a “modest step” toward using the cluster as one would use a single computer.

As an example, Yuan showed how a simple task (counting word frequency in a set of documents) could be expressed with the system. The LINQ expression “select many” is used to transform the set of documents into a set of words, then “group” is used to collate the words into groups. Finally, the count can be computed for each word. At each step, the use of LINQ’s relational algebra ensures that the task will parallelize well, while Dryad provides benefits such as data location transparency.

A recurring element of both the presentation and the question-and-answer period was comparing DryadLINQ to MapReduce, as is used by Google. Yuan offered Dryad as a more generalized framework for concurrent programming, showing how MapReduce could be implemented with three operations. He also posited that DryadLINQ provided a clear separation between the execution engine and the programming model, whereas in his view MapReduce conflates the properties of the two. This places unnecessary restrictions on both the execution engine and the programming model.

Yuan explained that the project was released internally and is being used in several projects. He described several lessons drawn from the effort, praising deep language integration, easy parallelism, and an integrated cross-platform environment. In describing future work, he proposed exploring the types of programs that could be created with this approach, asked how he could better generalize the programming model, and called for languages providing strong static typing in the datacenter.

Audience interest was strong for the paper, and session chair Marvin Theimer was forced to cut off discussion to keep the session on schedule. Eric Eide of the University of Utah began by asking what opportunities programmers have to control the compilation of their programs, for example optimizing for failure tolerance versus performance. In reply Yuan explained that some of these optimizations would need to be handed to the execution engine as a policy but that the problem has not yet been tackled directly by the compiler. Brad Chen from Google wanted to know whether the programming model facilitated good intuitive assumptions about performance from programmers. Similarly, Armando Fox wondered whether data mining and/or machine learning could give performance estimates. Yuan explained that they were still learning about the system and that people are actively working on understanding the performance characteristics. Modeling performance with machine learning is ongoing work. Yuan was also asked about support for continuous querying, and he thought that this would be an interesting area for future work.

- ***Everest: Scaling Down Peak Loads Through I/O Off-Loading***
  Dushyanth Narayanan, Austin Donnelly, Eno Thereska, Sameh Elnikety, and Antony Rowstron, Microsoft Research Cambridge, United Kingdom

Dushyanth Narayanan presented Everest, which uses I/O offloading to mitigate peak load conditions on storage servers. This work follows his paper at FAST earlier in the year, in which Dushyanth applied a similar mechanism to the problem of power consumption. He began by showing a trace of a production Exchange server; despite overprovisioning, an observed I/O load was shown to increases the server's response time twentyfold. At the same time, this load increase was not correlated with the workload on other disks. This key observation provides hope that spare resources on these other hosts can be utilized to lessen the burden on storage systems.

Everest operates within a client/server model. Any machine that needs protection against peak loads runs an Everest client, while a set of other machines operate Everest stores. In practice Everest clients may also be stores. Under normal operation, requests to an Everest client merely pass through to the local storage without modification. If the client detects that a peak load condition may be occurring (according to some threshold), it begins write offloading. This means preserving disk I/O bandwidth by issuing writes to Everest stores across the network interface. Once the peak subsides, the client stops offloading and begins reclaiming previously offloaded writes. Naturally, this solution targets temporary peaks in load; it will not provide long-term relief to a cluster that is fundamentally under-provisioned.

To minimize the burden of additional writes to the client's disk, the destination of offloaded writes is not written to local storage. Everest store nodes instead track the source of writes along with the data, written to a circular log. In recovering from a failure, clients query relevant stores to find all outstanding writes. Everest also includes features beyond this basic operation, such as offloading to multiple stores and load balancing, although these features were not discussed in depth.

Questions addressed a variety of motivational and technical details. Jonathon Duerig of the University of Utah gently challenged the assumption that there is no correlation between peak loads and asked whether there was a metric for how much correlation the system could endure. Dushyanth had not performed this analysis yet and was not aware of a synthetic workload that would allow experimenters to vary peak load correlation. Preston Crow of EMC asked whether disk caches could be grown to provide the same effect, but Dushyanth replied that the size of the observed peaks was in the gigabyte range—simply too large for a reasonably sized cache. Finally, Armando Fox of UC Berkeley wondered whether new failure modes were being introduced by spreading data across many volumes. Dushyanth thought that *N*-way replication, already present in Everest, was likely necessary to ensure that the system could tolerate failures in $N - 1$ Everest stores.

- ***Improving MapReduce Performance in Heterogeneous Environments***
  Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica, University of California, Berkeley

Matei Zaharia discussed the MapReduce programming model and the improvements his team at Berkeley has brought to Hadoop's implementation and algorithm. Their principal findings were based on the observation that performance heterogeneity breaks task-scheduling assumptions. Such heterogeneity is not hard to find in practice; for example, utility computing services don't generally provide strong performance isolation guarantees.

In the process of analyzing Hadoop, Matei used Amazon's EC2 service as an academic testbed, a direction he vigorously encouraged others to follow. In their experiments, they were able to operate at a convincingly large scale, yet with relatively low cost.

Backup selection in Hadoop occurs as nodes used for primary tasks become free. In that case, primary tasks that have not made sufficient progress will be replicated onto free nodes. The study found four principal problems with backups in Hadoop. First, too many backups will thrash network bandwidth and other shared resources. Second, the wrong tasks can be selected for backup. Third, backups can be directed to slow nodes, which is suboptimal. Fourth, if tasks are started at nonuniform times, scheduling decisions can be made incorrectly. In an example, Matei showed how 80% of the tasks for an operation could be selected for backup, with the majority of these tasks losing to the originals.

To fix these problems, a LATE scheduler was contributed to the Hadoop project (available at http://hadoop.apache.org/core/). The new scheduler estimates the completion time for each task and selects only slow-running tasks that are believed to complete late. It also caps the number of backup tasks and places backups on fast nodes. In the evaluation it was shown that this new scheduler offered a 58% performance improvement, on average.

Brad Chen of Google asked how tasks of variable length would be scheduled. In their work, Matei and his team had assumed that tasks were of roughly constant length. This is justified in that Hadoop itself attempts to maintain this invariant. To tackle this problem, Matei believed that one would probably need to prioritize according to task size; however, this problem has yet to be addressed. Marvin Theimer of Amazon also asked whether the authors had any advice for utility computing providers. The presenter encouraged more visibility into machine status, network topology, and rack locations.

OS ARCHITECTURE

Summarized by Dutch Meyer (dmeyer@cs.ubc.ca)

- ***Corey: An Operating System for Many Cores***
  Silas Boyd-Wickizer, Massachusetts Institute of Technology; Haibo Chen, Rong Chen, and Yandong Mao, Fudan University; Frans Kaashoek, Robert Morris, and Aleksey Pesterev, Massachusetts Institute of Technology; Lex Stein and Ming Wu, Microsoft Research Asia; Yuehua Dai, Xi'an Jiaotong University; Yang Zhang, Massachusetts Institute of Technology; Zheng Zhang, Microsoft Research Asia

Silas Boyd-Wickizer presented a new operating system interface as part of ongoing work on the Corey operating system. The work is premised on the observation that for application performance to scale with an increasing number of processing units, the time spent accessing kernel data structures must be made parallelizable. In current practice, synchronizing access to such data must be done conservatively. Although kernel subsystems can sometimes be redesigned to avoid or eliminate this locking, the process is time consuming for programmers. Such an approach, Silas argued, forces developers into a continuously incremental evolution of the kernel. As an alternative, Corey seeks to remove unnecessary sharing by letting applications explicitly declare when they intend kernel state to be shared.

Corey provides three new interfaces for creating kernel data structures: shares, address ranges, and kernel cores. However, only the first two were discussed in the presentation, each with an example. To demonstrate the use of shares, Silas showed how different file descriptors opened by different processes could result in poor performance scaling. Some access to the global file descriptor table was shown to be clearly contentious, with an order of magnitude decrease in performance. However, much of the synchronization around this table is clearly unnecessary, since processes may be using unrelated file descriptors. Corey's support for shares allows an application to explicitly define a private file descriptor to avoid the unnecessary locking. To demonstrate address ranges, Silas showed how shared memory can result in contention, even when processes are not actively writing into the shared memory space. The solution was to build a private memory space but to map in shared memory when explicitly requested by the application. This allows updates to private memory to occur without contention. Both examples were demonstrated with microbenchmarks. During the question-and-answer session Silas conceded that the workloads used in the system's evaluation targeted only the observed contention behavior, not the overall performance. He believed that overall performance would increase relatively with more cores and would likely be visible on a 32-core system.

The question-and-answer session was very active, and Silas's jovial nature drew appreciation from the audience on several occasions. Marvin Theimer of Amazon wondered whether any degree of public/private locking would prove to be scalable in a many-core architecture. He pointed out that successful grid and cluster architectures eschewed locking entirely, in favor of message passing. Silas pointed to the potential for nonglobal sharing in Corey as a possible means of scaling into a large number of cores. Alex Snoeren of UCSD pointed out that the architecture extracted the basic elements of NUMA and asked whether partitioning into sharing groups might be done automatically. Although this had been considered and may be a part of future work at MIT, Silas stressed that sharing explicitly could be a desirable quality to programmers.

- ***CuriOS: Improving Reliability through Operating System Structure***
  Francis M. David, Ellick M. Chan, Jeffrey C. Carlyle, and Roy H. Campbell, University of Illinois at Urbana-Champaign

Francis David addressed issues of system failure in his presentation on CuriOS. The system attempts to make use of restart-based recovery to deal with errors and uses a novel separation of client state to ensure that service restart does not result in additional failures.

Francis began by showing how faults in Minix 3, L4, Chorus, and Eros could each result in lingering failures even after a successful service restart. The problem is that a service stores state on behalf of its clients. When the service fails, its state is restarted along with the service, which puts clients in an inconsistent state. More broadly, the authors identified four requirements for transparent restart: address transparency, suspension of the client for recovery, and persistence and isolation of client state.

To provide these properties, Server State Regions (SSRs) were introduced. Each SSR represents the current state of a client and exists in an isolated address space. When a request is received by a server, the SSR is temporarily mapped into the server's address space and is correspondingly unmapped for the response. In the event of a fault, the server will only have access to the process it was directly servicing. On recovery, the client data can be checked for consistency. To test the techniques, a working timer and network service were created. These were validated with fault injection on an embedded platform. Faults, in this case, are memory aborts or bit flips that result in a system crash. With SSRs, CuriOS was able to recover from more than 87% of these failures and was usually able to restart the failed service. Francis evaluated the system with microbenchmarks, showing that the overhead consisted mainly of flushing the TLB and was therefore similar to a context switch.

Francis then fielded questions from the audience. Emin Gün Sirer from Cornell took a moment to disagree with an earlier claim that type safety would fix many faults in system operation, effectively strengthening the motivation for the work. He then asked whether the fault detection relied on a component detecting its own fault. When Francis replied that the fault detection had been ported from the existing code base, Gün asked whether fault detection needed to be protected in general. To this, Francis agreed and pointed to techniques for checking in-line, but he made it clear that their focus was exclusively on recovery, not detection. Francis was also asked about developing user-mode recovery services and recovering from faults in the SRR management service itself. Both were identified as areas for future work. He also clarified a few additional points from his presentation before session chair Remzi Arpaci-Dusseau was forced to cut off the discussion in the interests of time.

- ***Redline: First Class Support for Interactivity in Commodity Operating Systems***
  Ting Yang, Tongping Liu, and Emery D. Berger, University of Massachusetts Amherst; Scott F. Kaplan, Amherst College; J. Eliot B. Moss, University of Massachusetts Amherst

Ting Yang presented Redline, an operating system that provides first-class support for interactive applications. The commonly used schedulers in popular operating systems strive for a state of fairness. This ideal is one in which each process shares resources equally at fine granularity. The best-known alternative to this is real-time scheduling, in which processes receive dedicated resources to ensure timely responses. Interactive applications sit between these two ideals. They may often be idle for long periods of time followed by bursts of activity, as driven by an external event. Arguably, this can be a more important metric than fairness, since it is capable of capturing the user's perception of responsiveness.

To illustrate the challenge of scheduling for interactive applications Ting showed how simultaneous video playback and kernel compilation results in jumpy and unresponsive video playback. The problem is one of resource management. For example, both applications rely on getting data from disk, but the queues and caches in the storage stack are not necessarily fair.

The goal of Redline is to maintain responsiveness in applications that need it. It operates by coordinating resource management to devices as well as CPU and memory. Users identify interactive applications and reserve the desired CPU, memory, and IO priority. In the case of memory management, interactive applications are vulnerable to LRU eviction because they scan memory less frequently. Worse, as their working set shrinks, they fault more frequently, making the problem a potentially degenerative case. To correct this, Redline preserves working set size by protecting the pages of interactive applications for 30 minutes. If the system has insufficient memory for this assurance, processes will be degraded to best effort. At the same time, memory access by best-effort applications are "speed-bumped" to keep them from touching memory too frequently.

To evaluate Redline, Ting showed the result of video playback under the pressure of a fork bomb, a malloc bomb, and an IO intensive workload. In each, Redline was shown to outperform Linux (measured in frames per second) by a significant margin.

Ashvin Goel, from the University of Toronto, noted that the resource specification is described differently for each type of resource and wondered why a more unified approach wasn't used. Ting acknowledged that uniformity is desirable but explained that this approach was difficult in practice. They ultimately found that different descriptions were appropriate for each resource. To simplify the specifications, default rules and inference take some of the burden off the user. Ting also addressed concerns of starvation, stating that it might be necessary given that the goal was to provide isolation for interactive applications.

MONITORING

Summarized by Olga Irzak (oirzak@cs.utoronto.ca)

- ***Network Imprecision: A New Consistency Metric for Scalable Monitoring***
  Navendu Jain, Microsoft Research; Prince Mahajan and Dmitry Kit, University of Texas at Austin; Praveen Yalagandula, HP Labs; Mike Dahlin and Yin Zhang, University of Texas at Austin

Navendu Jain began by pointing out the importance of monitoring large-scale distributed systems. A motivating example showed that best effort is not always sufficient. For an application monitoring PlanetLab, half of the nodes' reports deviated by more than 30% from the true value, and 20% of the nodes by more than 65%. These inaccuracies are due to slow nodes, slow paths, and system reconfiguration. Hence, a central challenge in monitoring large-scale systems is safeguarding accuracy despite node and network failures. Network imprecision exposes the state of the network so that applications can decide whether or not to trust the accuracy of the result. Current techniques used for scalability in monitoring systems are aggregation, arithmetic filtering, and temporal batching. However, during network or node failure, these techniques can cause failure amplification, silent failure, and blockage of many updates even for a short disruption. As a result, the authors suggest accepting that the system is unreliable and quantify system stability using the "Network Imprecision" metric.

Instability manifests itself in either missed/delayed updates or double-counted updates. To quantify the former, a lower bound on the number of nodes whose recent inputs are guaranteed to be in the result (Nreachable) versus the number of total nodes in the system (Nall) is reported. The Ndupmetric is an upper bound on the number of nodes that are double-counted. Together, these metrics are useful to expose the impact of disruptions on monitoring accuracy. Network imprecision is application-independent, inexpensive, and flexible. Applications can set a policy to improve accuracy by applying the right techniques, such as filtering inconsistent results or performing redundant or on-demand reaggregation. Filtering with network imprecision results in 80% of the reports having less than 15% error, in contrast to best effort, in which there is 65% error. Network imprecision is simple to implement but difficult to implement efficiently. Using DHT trees, which form a butterfly network, intermediate results can be reused across different trees. This reduces the load from $O(Nd)$ to $O(d\log N)$ messages per node.

Discussion began with an observation and a challenge. Observation: Presented was a very nice subset of what DB people call observation quality—add metrics that are available where things such as freshness come into account. This gives more metrics to make decisions with. Challenge: Wouldn't it be nice if I could ask a monitoring system for a particular level of accuracy or network imprecision or whatever, and could you construct me a system that gives me that level of accuracy? The approach has been to separate the measurement mechanism from the policy that sets the accuracy bounds. Applications can accept results with a given accuracy bound and throw away others. But you can't guarantee an accuracy bound because of the CAP problem. Since the problem is closely related to aggregation techniques from sensor networks, the next questioner wondered whether looking at order- and duplicate-insensitive aggregation functions would be in order. These are complementary because in principle these try to minimize the impact of disruptions on accuracy, whereas network imprecision exposes disruptions. The SM techniques provide a policy that can be applied in this framework.

- ***Lightweight, High-Resolution Monitoring for Troubleshooting Production Systems***
  Sapan Bhatia, Princeton University; Abhishek Kumar, Google Inc.; Marc E. Fiuczynski and Larry Peterson, Princeton University

Sapan Bhatia made the observation that there will always be bugs in production systems despite analysis and testing frameworks. The problem is aggravated as system complexity increases. There are easy and hard bugs in systems. Easy bugs come with an exact description and are easy to reproduce. Hard bugs are hard to characterize, hard to reproduce, hard to trace to a root cause, spatially ambiguous, temporally removed from root cause, and can be intermittent and unpredictable. The authors experienced many hard bugs in their dealings with PlanetLab, such as intermittent kernel crashes with an out-of-memory bug, unusual ping latencies, and a kernel crash every 1–7 days (or not). A medical analogy is that easy problems are single, localized injuries such as a broken foot, whereas a hard one might be a vague sense of malaise. To facilitate handling such bugs, the authors present Chopstix. Chopstix monitors low-level events (vital signs) such as scheduling, I/O, system calls, memory allocation, and cache misses. It then captures abnormal deviations in the system's behavior; these are referred to as symptoms. As logging all events is too expensive, sampling is used. Uniform sampling is biased toward high-frequency events. Instead, Chopstix uses frequency-dependent sampling from the measurement community, which uses a sketch—an approximation of the frequency distribution of a set of events. On an event trigger, one extracts a signature of the event, hashes it, and updates the sketch. If a sampling decision is made, one collects the sample and performs logging. Collecting the sample is heavyweight but is more efficient because of the principle of locality. At the end of every data-collecting epoch (60 s in Chopstix), the collected data is transferred to user space and stored as a series of summaries. This results in a rich, fine-grained data set for an overhead of 1% CPU utilization and around 50 MB per day of disk consumption on each node.

The Chopstix GUI was also presented. The GUI polls events from different nodes for the requested vital signs. These are displayed on a graph that can be drilled into to see more detail. The GUI allows the use of intuition for certain bugs (e.g., high CPU usage means a busy loop). Alternatively, use of a library of diagnosis rules to detect less obvious conditions such as low process CPU utilization with high scheduling delay and high CPU utilization per system likely means a kernel bottleneck. An example of using Chopstix to solve an elusive PlanetLab bug was then presented. The observed behavior was that some nodes would freeze every few days (or not), there was no info on console, the SSH session would stall prior to hangs, and vmstat reported high IO utilization. Chopstix showed abnormal blocking and I/O vital signs from the journaling daemon, which turned out not to be responsible for the freezes. It also showed spikes in the scheduling delay coincident with spikes in kernel CPU utilization, which pointed to the critical bug in a loop in the scheduler. They evaluated performance overhead using the lmbench microbenchmark. The slowdown is between 0% and 2.6% for getpid. Kernel compile and Apache macrobenchmarks showed almost no overhead in the benchmarks.

The probability of an error is a function of the distribution of counters in a sketch. By varying the size of the sketch, the probability of error is between 0.0001% and 0.00001%. Future work includes automating bug detection by data-mining Chopstix data and combining NetFlow and Chopstix data to diagnose network-wide behavior.

The first questioner asked whether Bhatia could compare work with DCPI from SOSP 1997. They were also monitoring with 1.2% overhead. Bhatia replied that they had a simpler notion of sampling. Their work is similar to oprofile, which this beats. Another person wondered about diagnosis rules. Is it possible (looking into the future) to capture the thinking of a systems guru looking at this output by data-mining? There are various knowledge-base systems that analyze code for bugs, but they are imperfect, so you really need human expertise to analyze rich data. Someone else pointed out that this is useful for the developer who is a kernel hacker and knows the system inside out, but how could it be extended to abstract the vital signs to a higher level and give signals of more abstract health, so that a system could say, "I'm sick, I need to see a doctor"? A lot of intelligence could be coded into rules, and the rule database could grow over time (comparable with WebMD), allowing diagnosis by less experienced people. How scalable can this be made if you want hundreds or thousands of metrics? Bhatia said that they haven't carried out these experiments. The <1% utilization is between 0.1% and 1%, so there should be more room to squeeze in other metrics. Finally, someone pointed out an issue: In the 1970s and 1980s medical diagnosis with AI was prevalent, but there was an explosion after 20 or so rules, so there was a move to other systems such as Bayesian filtering. One of the reviewers was excited about rule-based systems and wanted it cited, but it was hard to find. Maybe that should tell us something.

- ***Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions***
  Xu Chen, University of Michigan; Ming Zhang, Microsoft Research; Z. Morley Mao, University of Michigan; Paramvir Bahl, Microsoft Research

Xu Chen highlighted the fact that enterprise networks support various business-critical applications such as VoIP and email. In a large network, there are usually thousands of applications running simultaneously, thousands of people are doing IT support, and lots of money is spent thereon. Network management is complicated since applications are very complicated and distributed across multiple components (e.g., MS Office Communicator, a VoIP/messenger app, uses DNS, Kerberos, VoIP, Director, and many back-end servers such as file and SQL). Extracting dependency information is hard. Applications are heterogeneous, in terms of functionality and deployment. The knowledge of these dependencies is distributed across layers and locations. Also, applications continuously evolve, adding new services periodically and reconfiguring/consolidating others. Currently, when there is a service outage, human knowledge and understanding of the system and its dependencies is used to troubleshoot the problem. This is expensive, error-prone, and difficult to keep up-to-date. This provides the motivation for automatically discovering dependencies for network applications.

This work introduces a new technique to discover service dependencies based on delay distributions, identifies the limitations of dependency discovery based on temporal analysis, and evaluates the technique on five dominant applications in the Microsoft enterprise. The goal is to design a generic solution for various applications using nonintrusive packet sniffing and TCP/IP header parsing. The proposed system, called Orion, identifies the time delay between dependent services, which reflects the typical processing and network delay. Orion identifies service based on IP address, port, and protocol. To make this scalable, ephemeral ports are ignored. Another problem is that dependencies exist between application messages. To address this Orion analyzes only TCP/IP headers and aggregates packets into flows. This reduces bias introduced by long flows and reduces the number of pairs. Orion needs a fair number of samples to infer dependencies. That can be overcome by aggregating across clients, servers and ports.

Orion was deployed in an MS enterprise network. It focused on extracting dependencies from MS Exchange, Office Communicator, Source Depot (such as CVS), Distributed File System, and intranet sites. Orion has no false negatives and a smaller footprint than previous work. As we see more flows, the false positives increase, but false negatives decrease. True positives eventually converge to the correct value. Orion's drawback is that it requires training, isn't applicable to P2P applications, and may miss certain kinds of interactions (e.g., periodic ones). It may include false positives. Lessons learned are that temporal analysis is limited, as it has no app-specific knowledge. Regardless, false positive can be reduced to a manageable level.

Why did filtering induce a small peak in the delay distribution in the rightmost bins? Chen said that this was an artifact of the filtering, but it doesn't affect the result. It may need more sophisticated FFT. Someone else pointed out that in a datacenter, you may have services dependent on services and so on, so a client action may go through lots of servers indirectly, so is the resolution enough to identify this without deep packet inspection? Chen said you could definitely do deep packet inspection, but temporal analysis can be made usable.

Finally, someone asked, given the prevalence of virtualization technology, services could be migrated among machines, so does the technique take that into account? Service migration is a potential application of this information, allowing dependent servers to migrate together.

WORK-IN-PROGRESS REPORTS (WIPS)

Summarized by Daniel Peek (dpeek@eecs.umich.edu)

- ***Multikernel: An Architecture for Scalable Multi-core Operating Systems***
  Simon Peter, Adrian Schupbach, Akhilesh Singhania, Andrew Baumann, and Timothy Roscoe, ETH Zurich; Paul Barham and Rebecca Isaacs, Microsoft Research, Cambridge

Locking and shared data limit scalability, so we should treat multiple cores in a computer as if they are on a network and use distributed systems ideas to get them to work together. These include partitioning, replication, and agreement protocols. For more information see www.barrelfish.org.

- ***Transcendent Memory: Re-inventing Physical Memory Management in a Virtualized Environment***
  Dan Magenheimer, Chris Mason, David McCracken, and Kurt Hackel, Oracle Corporation

Efficient memory utilization in the presence of virtual machines (VMs) was addressed. As an improvement on ballooning techniques, idle memory held by VMs can be reclaimed. VMs can access the resulting pool of memory through paravirtualized APIs.

- ***Towards Less Downtime of Commodity Operating Systems: Reboots with Virtualization Technology***
  Hiroshi Yamada and Kenji Kono, Keio University

This project explores the use of virtual machines to improve the speed of reboots caused by patching. To patch a system, first a VM clone of the currently running system is created. The clone is patched and rebooted. Then, the clone replaces the currently running system. This avoids disruption of the currently running system until the clone system is patched and rebooted.

- ***TeXen: Virtualization for HTM-aware Guest OSes***
  Christopher Rossbach, UT Austin

TeXen is the first virtual machine monitor to use hardware transactional memory to virtualize HTM-aware OSes. This combination has difficulties, such as preserving the guarantees provided by HTM hardware, and opportunities, such as moving the complexity of input and output out of the kernel.

- ***SnowFlock: Cloning VMs in the Cloud***
  H. Andres Lagar-Cavilla, University of Toronto

VMs in cloud computing need to be improved. They can take minutes to start up and require application-specific state to be transferred to them. Instead, we should be able to use a fork-like interface to quickly create many stateful VMs.

- ***CPU Scheduling for Flexible Differentiated Services in Cloud Computing***
  Gunho Leo, UC Berkeley; Byung-Gon Chun, Intel Research Berkeley; Randy H. Katz, UC Berkeley

No summary available.

- ***Toward Differentiated Services for Data Centers***
  Tung Nguyen, Anthony Cutway, and Weisong Shi, Wayne State University

Because application demands differ, developers should be able to choose VMs with varying properties, such as the number of replicas and network topology. Instead of offering generic VMs, cloud computing providers can offer several kinds of VMs, each with varying properties.

- ***TCP Incast Throughput Collapse in Internet Datacenters***
  Yanpei Chen, Junda Liu, Bin Dai, Rean Griffith, Randy H. Katz, and Scott Shenker, University of California, Berkeley

In a situation with *N* clients connected to a server through a switch, application throughput drops dramatically when all of the clients attempt to communicate simultaneously. This work explores the interaction of this communication pattern and TCP and proposes changes to TCP to improve this situation. Although simply adding more buffer space on the switch would mitigate this problem, the underlying TCP issues should really be solved by a fix to TCP.

- ***Gridmix: A Tool for Hadoop Performance Benchmarking***
  Runping Qi, Owen O'Malley, Chris Douglas, Eric Baldeschwieler, Mac Yang, and Arun C. Murthy, Yahoo! Inc.

Gridmix is a set of Hadoop benchmarks that addresses the needs of several audiences, including Hadoop developers, application developers, and cluster builders.

- ***CrystalBall: Predicting and Preventing Inconsistencies in Deployed Distributed Systems***
  Masyam Yabandeh, Niola Knezevic, Dejan Kostic, and Victor Kuncak, EPFL

Many errors in distributed systems are a result of violations of safety properties. To avoid these situations, members of a distributed system can gather snapshots of neighboring participants and do state space exploration to understand the results of actions and steer execution away from inconsistent states.

- ***Coscheduling of I/O and Computation in Active Storage Clusters***
  Saba Sehrish, Grant Mackey, and Jun Wang, University of Central Florida

This work deals with Hadoop map tasks that work with several data objects that may not be co-located. The authors are making a scheduler for map tasks that takes into account the performance difference between local and remote I/O operations.

- ***Honor: A Serializing On-Disk Writeback Buffer***
  Rick Spillane, Chaitanya Yalamanchili, Sachin Gaikwad, Manjunath Chinni, and Erez Zadok, Stony Brook University

Random writes are becoming a larger component of many I/O workloads, but for performance reasons sequential writes are preferable. This project redirects writes to a separate disk, called the sorting disk, to sequentially log writes. These logged writes can later be applied to a general file system.

- ***Zeno: Eventually Consistent Byzantine Fault Tolerance***
  Atul Singh, MPI-SWS/Rice University; Pedro Fonseca, MPI-SWS; Petr Kuznetsov, TU-Berlin/T-Labs; Rodrigo Rodrigues, MPI-SWS; Petros Maniatis, Intel Research Berkeley

Storage backends generally favor availability over consistency. This project proposes an eventually consistent byzantine-fault-tolerance algorithm to improve consistency in such storage systems.

- ***Scalable Fault Tolerance through Byzantine Locking***
  James Hendricks and Gregory R. Ganger, Carnegie Mellon University; Michael K. Reiter, University of North Carolina at Chapel Hill

To improve throughput and latency of byzantine-fault-tolerant systems, this project allows clients to use a byzantine-fault-tolerance algorithm to acquire a byzantine lock on a part of the system state and specify the order of operations.

- ***Fault Tolerance for Free***
  Taylor L. Riche and Allen Clement, The University of Texas at Austin

Multicore machines are now widely available; the difficulty is in programming fault tolerance for these systems. This project takes programs that are already constructed in a language that maps applications to multicore machines and reuses those interfaces to provide redundant execution for fault tolerance.

- ***Writing Device Drivers Considered Harmful***
  Leonid Ryzhyk and Ihor Kuz, University of New South Wales

The information needed to make device drivers is in the specification of the OS interface and the device interface. This work automatically creates drivers by composing state machines representing both specifications.

- ***CitySense: An Urban-Scale Open Wireless Sensor Testbed***
  Ian Rose, Matthew Tierney, Geoffrey Mainland, Rohan Murty, and Matt Welsh, Harvard University

This project aims to build a 100-node city-wide sensor network testbed aimed at public health studies, security, and novel distributed applications. Researchers can get an account to run experiments. See www.citysense.net for further details.

- ***WiFi-Reports: Improving Wireless Network Selection with Collaboration***
  Jeffrey Pang and Srinivasan Seshan, Carnegie Mellon University; Ben Greenstein, Intel Research Seattle; Michael Kaminsky, Intel Research Pittsburgh; Damon McCoy, University of Colorado

WiFi-Reports aggregates user-contributed information about the quality of pay-to-access wireless access points. Challenges include privacy, fraud, and estimation of packet loss regimes with distributed measurements.

- ***S3: Securing Sensitive Stuff***
  Sachin Katti and Andrey Ermolinskiy, University of California, Berkeley; Martin Casado, Stanford University; Scott Shenker, University of California, Berkeley; Hari Balakrishnan, Massachusetts Institute of Technology

This project aims to prevent high-bandwidth data theft by external attackers by enforcing policies at data egress points such as the network and USB keys. A network of hypervisors is used to track information flow at the word level. This uses hardware support for virtualization and speculation for performance.

- ***Communities as a First-class Abstraction for Information Sharing***
  Alan Mislove, MPI-SWS/Rice University; James Stewart, Krishna Gummadi, and Peter Druschel, MPI-SWS

A community is a densely connected subgraph of users in an online service (e.g., Facebook, MySpace). These communities can be leveraged to infer trust, control access to communities, and discover more relevant search results.

FILE SYSTEMS

Summarized by Vivek Lakshmanan (vivekl@cs.toronto.edu)

- ***SQCK: A Declarative File System Checker***
  Haryadi S. Gunawi, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

Since filesystem corruption can lead to data loss, it is important to keep filesystems consistent at all times. Nonetheless, filesystems continue to be corrupted. Although techniques such as journaling have proved to be effective against the most common cause for filesystem inconsistency, system failure during metadata updates—it is not sufficient to repair filesystem corruption. Offline filesystem consistency checkers such as fsck are therefore seen as the last line of defense against data loss. However, commodity filesystem consistency checkers (e.g., e2fsck for ext2) can themselves corrupt the file system. The authors suggest that the implementation complexity of consistency checks—normally written in C—is a major contributor to the limited reliability of such tools. SQCK improves filesystem reliability by providing a SQL-based declarative language to filesystem developers to succinctly define consistency checks and repairs. The result is the ability to encode the functionality of 16 KLOC from e2fsck in 150 SQL queries.

Type-aware fault injection was used to expose a number of flaws in e2fsck. For instance, corrupting an inode's indirect pointer to point to the superblock results in an unmountable filesystem when processed by e2fsck. In addition, e2fsck does not take advantage of all the information available to it during repairs, making consistent but incorrect repairs. SQCK decouples specification of consistency checks and repair policies from the interpretation of filesystem structure, making it easy to specify intent. SQCK takes an FS image and loads all of its metadata into database tables. The specified SQL queries are run on these tables and any modifications are then serialized back to the filesystem image. Several examples of existing fsck checks and their equivalent in SQCK to illustrate the improvement were presented. When specifying filesystem repairs declaratively in SQL is not possible, a library of SQL queries can be composed together through bits of C code to produce the desired effect.

The current prototype uses a MySQL database that holds the tables in a RAM disk. The 150 queries representing e2fsck were implemented in 1100 lines of SQL. It is also easy to add new checks or repairs. Through the use of some performance optimizations, the runtime overhead on their current prototype was brought within 50% of the existing e2fsck.

A questioner asked about the size of the RAM disk for e2fsck. Gunawi explained that the database only concerns itself with metadata. For a half-full 1-TB filesystem, roughly 500 MB of storage is required for metadata. Another attendee wondered whether the comparison of the number of lines of code between e2fsck and SQCK took the complexity of the scanner-loader that interprets the filesystem image. Gunawi said that e2fsck includes 14 KLOC of scanning code, which was ignored in the comparison. When asked whether reordering SQL statements can affect correctness, the presenter said that an automated reordering of the SQL statements has not been implemented, but he found that ensuring the ordering manually was easier in SQCK. An attendee wondered whether flushing back to disk could introduce errors. Gunawi said that e2fsck itself does not atomically repair the filesystem; thus a crash during e2fsck could cause another inconsistent state. However, implementing transactional updates in SQCK would be simpler, although the current implementation does not have such support. The final issue raised was whether it wouldn't be easier to specify what the filesystem should look like than to specify checks. Gunawi explained that this was indeed the original intent and led them to explore developing their own declarative language. However, they found that SQL was a close fit and already had traction in the technical community.

- ***Transactional Flash***
  Vijayan Prabhakaran, Thomas L. Rodeheffer, and Lidong Zhou, Microsoft Research, Silicon Valley

Solid state disks (SSDs) are a significant departure from traditional magnetic disks. Currently, these disks have adopted the existing thin interface with which existing magnetic disks comply. Although this allows backward compatibility, Vijayan states that this choice represents a lost opportunity in proliferating new abstractions to truly take advantage of this shift in storage technology. TxFlash is a new abstraction that allows SSDs to provide transactional semantics. TxFlash proposes transactional support in SSDs, reducing file system complexity while providing performance improvements. The existing logic in SSDs to handle write-leveling and cleaning makes them particularly well suited for transactional support.

A commit protocol is essential to provide transactional semantics. The traditional log-based commit protocol is heavily used for its relative simplicity: A separate log is maintained where data is written, following which an explicit commit record is written. Once committed, write-back can occur asynchronously. This adds a space overhead for the explicit commit record, as well as a performance penalty owing to the strict ordering required between the data and commit records. These can have a significant impact on workloads that require short and frequent transactions. TxFlash's new WriteAtomic and Abort abstractions could eliminate these problems. The WriteAtomic abstraction can inform the TxFlash device what pages need to be updated atomically. The TxFlash device links pages committed within the same transaction to form a closed loop, which can then be written in parallel. A commit is said to be successful if the updates cause a cycle. However, broken cycles are not a sufficient condition for detecting aborted transactions. To solve this ambiguity, the Simple Cyclic Commit (SCC) protocol is proposed by the authors. It ensures that if a version of a page exists, any previous version of the same page on disk must be committed, and if a transaction aborts, the affected pages and their references are erased before a newer version is retried.

TxFlash was evaluated through a simulator as well as an emulator implemented as a pseudo block device. The authors also implemented a modified version of the ext3 file system, TxExt3, which exercises the WriteAtomic interface. This approach allowed the authors to strip out roughly 50% of the ext3's journaling implementation. Benchmarks suggest that TxFlash improves performance by roughly 65% for IO-bound workloads, particularly for those with small transactions, while having a negligible impact on performance for large transactions.

An audience member asked what deployment scenarios the authors envisioned that would proliferate the use of abstractions like those proposed in TxFlash. Vijayan said that reduced complexity of the software stack makes TxFlash particularly interesting for embedded devices and specialized operating systems such as those in game consoles. Another attendee asked whether there was a way to implement transactions on existing interfaces. How about exposing a fuller view of flash including metadata to software? Vijayan believes that it is not sufficient to expose the metadata. One would either have to export all the functionality to the software or implement it in the disk. The latter seems preferable to hide the complexity from software.

- ***Avoiding File System Micromanagement with Range Writes***
  Ashok Anand and Sayandeep Sen, University of Wisconsin, Madison; Andrew Krioukov, University of California, Berkeley; Florentina Popovici, Google; Aditya Akella, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, and Suman Banerjee, University of Wisconsin, Madison

The modern disk interface is represented by a linear address space; this aids usability but limits the opportunity for the software stack to make performance improvements. File systems try to improve disk I/O by ensuring spatial locality of related blocks; however, the interface only allows them a low-level command to write a block to a specific logical address—a classic example of micromanagement. The information gap between the file system and the disk results in I/O operations incurring unnecessary positioning overhead. This work proposes a new interface where the file system can specify a range of candidate blocks to the disk and allows it to choose the most appropriate block.

The interface allows the file system to specify a list of ranges to the disk, which, in return, specifies the result of the request as well as the chosen target block for the write request. One possible problem is that file systems might specify multiple write requests that have overlapping ranges. If the disk selects a block from the overlapping range, then it must make sure it masks it out of the range that it considers for the subsequent write request. This additional metadata must eventually be cleared. An ideal opportunity to do so is during write barriers.

This change in interface would require the in-disk scheduler to consider all options from among the list of ranges passed to it by the OS but select only one. A simple modification is the expand-and-cancel (ECS) approach. It expands a range write into a number of write requests. Once a candidate is selected, all other ranges are canceled from the request queue. This, however, is computationally expensive, since extensive queue reshuffling may be required. The authors present an alternative called Hierarchical Range Scheduling (HRS) in their paper.

Integration into existing file systems also poses a challenge. For instance, file systems try to improve sequentiality of writes. However, with range writes, related blocks could be spread farther apart. Moreover, there are complications because the file system is informed of the chosen destination later than normal. However, the presenter suggested that these were not significant impediments to using range writes, at least for a subset of block types. Simulation runs of range writes on write-heavy workloads such as Postmark and untar showed that the performance improved by 16%–35%.

An audience member wondered whether it was sufficient to create a faster write cache without modifying the filesystem, disk, and interface to improve write performance. The presenter replied that existing disks could benefit from their approach. Moreover, cooperative storage between the cache device and the backing store was also possible. Another audience member commented that if a high-speed write cache (e.g., a flash drive) was used, it is likely that the backing disks would be slow and inexpensive; therefore cooperation between the cache and backing disks through range writes might not be possible. Lastly, a few people wondered how range-writes would handle overwrites and if this approach wouldn't result in greater fragmentation. The presenter suggested that read and rewrite performance might suffer from fragmentation. He mentioned that they had not sufficiently evaluated this and that it was an area for future work.

PROGRAMMING LANGUAGE TECHNIQUES

Summarized by Vivek Lakshmanan (vivekl@cs.toronto.edu)

- ***Binary Translation Using Peephole Superoptimizers***
  Sorav Bansal and Alex Aiken, Stanford University

Sorav Bansal explained that binary translators are used to run applications compiled for one ISA (Instruction Set Architecture) on another. Major challenges in binary translation are performance, ISA coverage, retargetability, and compatibility with the OS. This work explores using superoptimization to address the first three of these challenges associated with binary translators. Superoptimization is an approach in which the code generator does a brute-force search for optimal code. The peephole superoptimization approach presented here utilizes three modules: a harvester, which extracts canonicalized target instruction sequences from a set of training programs; an enumerator, which enumerates all possible instructions up to a certain length, which are checked for equivalence against the target sequences produced by the harvester; and, finally, a rule generator, which creates a mapping of target sequences to their equivalent optimal translations.

Using peephole superoptimization for binary translation complicates matters. First a register mapping is required between the source and destination architectures before equivalence can be verified. The register mapping may need to be changed from one code point to another and the choice of the register mapping can have a direct impact on the quality of translation. The authors used dynamic programming to attempt to reach a near-optimal solution. Their mapping accounts for translations spanning multiple instructions and simultaneously performs instruction selection and register mapping.

A static user-level translator from ELF 32-bit PPC/Linux to ELF 32-bit x86/Linux binaries was implemented. The prototype was evaluated using microbenchmarks and SPECINT 2000 against natively compiled equivalents, as well as other binary translators such as Rosetta and QEMU. The minimum performance of the prototype was 61% of native, but it exceeded native on three microbenchmarks. It is up to 12% faster than Rosetta and 1.3 to 4 times better than QEMU. Note, however, that both Rosetta and QEMU are dynamic translators, so the cost of translation is embedded in their runtimes while their current prototype is static.

One attendee asked about the pattern-matching algorithm used and whether optimizing this would improve performance. The response was that the current implementation used a very simple approach and a pattern-matching algorithm could help. There were questions raised about whether the translator could work on multithreaded applications and whether the use of synchronization constructs was tested. The presenter didn't know of any reason why their approach wouldn't work for multithreaded applications. Correctness on synchronization constructs had not been tested. Another asked how well their approach would translate to a dynamic translator. Sorav explained that the choice of a static translator was for simplicity. However, he wasn't aware of any reason why the approach couldn't work in a dynamic setting. The final query concerned how well the equivalence test handled aliased instructions. Soral claimed that the Boolean filter used by the satisfiability solver was resilient to aliasing. Details of this can be seen in their previous publication in ASPLOS.

- ***R2: An Application-Level Kernel for Record and Replay***
  Zhenyu Guo, Microsoft Research Asia; Xi Wang, Tsinghua University; Jian Tang and Xuezheng Liu, Microsoft Research Asia; Zhilei Xu, Tsinghua University; Ming Wu, Microsoft Research Asia; M. Frans Kaashoek, MIT CSAIL; Zheng Zhang, Microsoft Research Asia

Zhenyu Guo described R2 as a record and replay tool for debugging. Some bugs are hard to reproduce by simply reexecuting the program, whereas others make it hard to do comprehensive analysis without significant perturbation to the system at runtime. Previous approaches have used virtual machines to replay the application and the OS, but this is difficult to deploy. Other attempts that use a library-based approach do not support multithreading or asynchronous I/O. Instead, R2 allows developers to determine the interface at which record and replay will occur and tries to address these shortcomings.

Selecting a replayable set of functions with the goal of capturing nondeterminism requires the developer to make a cut through the call graph. Calls to functions above the cut are replayed but functions below the cut are not. R2 also establishes an additional isolation environment: The items above the call graph cut are termed to be in replay space, whereas the R2 library and the rest of the system are in system space. Since it is possible to choose cuts poorly, the authors propose two rules developers can follow: Rule 1 (isolation)—all instances of unrecorded reads and writes to a variable should be either all below or all above the interface; Rule 2 (nondeterminism)—any nondeterminism should be below the interposed interface.

A major challenge is to keep a deterministic memory footprint between record and replay. It is important to get the same memory addresses in both record and replay to ensure that different control flow isn't followed. R2 uses separate memory allocators for system and replay space. Allocation in replay space will get similar addresses, whereas allocations in system space are hidden from the application and don't need to get a deterministic address. Another issue is ensuring deterministic execution order in multithreaded applications. It is important to make sure that the ordering of execution of multithreaded applications is not changed between the original and replayed runs. To do this, R2 captures happens-before relationships through causality events that have designated annotations.

There are three categories of annotations in R2: data transfer, execution order, and optimization. The authors have annotated large parts of the Win32 API, in addition to all of MPI and SQLite APIs. R2 has been used to debug several large applications with moderate annotation effort. A recording overhead of 9% was measured on ApacheBench when run on a standard Apache Web server configuration.

One audience member asked whether there were plans for any static checking or automatic verification for choosing the interface cut to prevent developers from making mistakes. The presenter stated that cutting at library boundaries was a good rule of thumb. However, work on using compiler techniques to select this boundary is planned. Another attendee wondered whether the tool could be used in production for more complex workloads, since the overhead for microbenchmarks is helped by caching. The presenter claimed that they saw 10%–20% overhead for larger benchmarks. He also said that one problem with larger applications is that the log may become unmanageable; as a result, a form of checkpointing is being considered. A query regarding R2's ability to cope with data races was posed. The presenter admitted that R2 would not be able to handle such issues currently. Finally, an audience member asked whether developers had to reannotate new versions of libraries. The presenter said that API-level annotation was sufficient.

- ***KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs***
  Cristian Cadar, Daniel Dunbar, and Dawson Engler, Stanford University

  Winner of Jay Lepreau Award for one Best Paper

Cristian Cadar began by saying that systems programming is notorious for its complex dependencies, convoluted control flow, and liberal unsafe pointer operations. The ever-present threat of malicious users does not make matters easier. Testing such complex code is not trivial. Certain bugs are only tripped in obscure edge conditions, which may be missed even when comprehensive test cases are available. KLEE is designed to check such complex systems code. KLEE is a fully automatic symbolic execution tool that explores a large number of paths in a program and automatically generates test cases.

KLEE runs C programs on unconstrained symbolic input. As the program runs, constraints on the symbolic values are generated; these are then fed to a constraint solver that generates test cases. Though this seems conceptually simple, there are a few major challenges KLEE overcomes. First, most programs have an exponential number of execution paths they may take. As a result, smart search heuristics are essential. KLEE uses either random path selection or coverage-optimized search at any given point (chosen in round-robin fashion): The former prevents starvation of certain paths, whereas the latter tries to choose the path closest to an uncovered instruction. The second challenge relates to constraint solving. Since KLEE needs to invoke a constraint solver at every branch, the costs can easily become prohibitive. As a result, KLEE tries to eliminate irrelevant constraints before sending them to the constraint solver. In addition, results from the constraint solver for previous branches are cached and reused where possible. Finally, when a program being executed by KLEE reads from the environment (filesystem, network, etc.), all possible values for the operation should be ideally available for KLEE to explore. Similarly, when the program writes to the environment, subsequent reads should reflect that write. To handle this in the case of symbolic input to a system call, KLEE provides the ability to redirect environmental accesses to models that generate the necessary constraints based on the semantics of the system call invoked.

KLEE was run on each of the 89 standalone applications, combining to form Coreutils for 1 hour. The test cases generated were then run on the unmodified tools under a replay driver that recreates the environment for the tests to proceed. Overall, KLEE's tests produced high line coverage, with an overall average of 84%. In comparison, manual testing achieves an average of 68% coverage. KLEE was able to find 10 unique crash bugs in Coreutils, each accompanied by commands that could reproduce crashes. It has been run on hundreds of applications as well as Minix and HiStar, and it has found 56 serious bugs in total.

An audience member wondered how KLEE would work against large systems such as database servers. The presenter replied that their previous work used symbolic file systems. Something similar could be done for databases. Another question involved the location of the bottleneck for KLEE when scaling the complexity of the checked system. The authors believe the number of states is a major problem. KLEE uses COW to minimize memory footprint. An audience member noticed that bullet-proof code could be a problem (last-minute checks in the code that return incorrect values but don't let the program crash). The presenter believed that the assert framework could be of help in such situations. in response to a question about the path lengths for some of the 56 bugs found, Cristian explained that the depth varied, but in some cases hundreds of branches needed to be hit. Another audience member asked whether they had attempted to use some of the tools from the related work to see if they could catch any of the bugs as well. Cristian replied that since many of the tools were not available or not comparable to their approach, they did not perform a comparison. The last question was regarding the name KLEE and what it meant. Cristian credited Daniel Dunbar with the name and believed that perhaps it was a Dutch word, a theory swiftly put to rest by a Dutch member of the audience.

SECURITY

Summarized by Periklis Akritidis (pa280@cl.cam.ac.uk)

- ***Hardware Enforcement of Application Security Policies Using Tagged Memory***
  Nickolai Zeldovich, Massachusetts Institute of Technology; Hari Kannan, Michael Dalton, and Christos Kozyrakis, Stanford University

Hari Kannan started by observing that traditional operating systems' lack of good security abstractions forces applications to build and manage their own security mechanisms. This bloats the TCB and makes it hard to eliminate bugs. But recent work has shown that application policies can be expressed in terms of information flow restrictions and enforced in an OS kernel, and this work explored how hardware support can further facilitate this.

Unfortunately, current hardware mechanisms are too coarse-grained to protect individual kernel data structure fields, so Hari proposed using tagged memory, where each physical memory word maps to a tag and tags map to access permissions. A trusted monitor below the OS is responsible for mapping labels expressing application security policies to tags that are enforced by the hardware, so less software needs to be trusted and some level of security is maintained even with a compromised OS.

Hari presented LoStar, a prototype system based on HiStar OS and a tagged memory hardware architecture called Loki. He discussed the operation and implementation of LoStar, including optimizations such as a multi-granular tag storage scheme to reduce memory overhead and a permission cache within the processor. The prototype had a 19% logic overhead (which would be much less for a modern CPU), had negligible performance overhead, and reduced the (already small) TCB of HiStar by a factor of 2. Hari concluded his presentation with pointers to the Loki port to a Xilinx XUP that costs $300 for academics and $1500 for industry, adding that the full RTL and the ported HiStar distribution are available.

A member of the audience asked Hari to contrast Loki with Mondrian memory protection, which also offers fine granularity. In particular, could LoStar be implemented on top of Mondrian memory protection or is there some fundamental functional difference? Hari acknowledged that Loki builds on Mondrian memory protection but argued that Loki provides more functionality and that the whole system is a carefully designed ecosystem. LoStar extends application security policies all the way to the hardware, and Loki keeps the MMU outside the TCB. Mondrian memory protection, in contrast, only extends traditional access control to offer word granularity and, furthermore, depends on the correctness of the MMU. Finally, in response to another question, he clarified that a dedicated garbage collection domain described in his presentation does not have unconditional access to kernel objects.

- ***Device Driver Safety through a Reference Validation Mechanism***
  Dan Williams, Patrick Reynolds, Kevin Walsh, Emin Gün Sirer, and Fred B. Schneider, Cornell University

Patrick Reynolds argued that device drivers should not be trusted, because they have a high fault rate and are written by third parties. These issues, in combination with running in the kernel and having hardware privileges, can lead to Trojan horses, insider attacks, and faults that take down the system. He said that Cornell University is building a new operating system, the Nexus, that is based on a trusted microkernel that leverages trusted hardware to enable trustworthy applications. Nexus aims for a small and auditable TCB, but device drivers presented a great challenge because they are so untrustworthy and keep changing.

Nexus moves drivers to user space as a first step to prevent direct attacks on the kernel, but devices can also compromise the kernel integrity in many ways, such as overwhelming the kernel with interrupts or using direct memory access. Nexus addresses this by placing a filter between the driver and the device. This filter is written once for each device, is small to audit, and, given proper specifications, can defend against broader attacks.

Direct hardware operations in drivers ported to Nexus are replaced with Nexus system calls and the sequence of operations is constrained using a specification language built around state machines. Illegal transitions kill the driver and invoke a trusted reset routine that is part of the specification. Several mechanisms, including IO ports, MMIO, interrupts, and DMA, are captured in the specification language. Finally, drivers are not trusted with disabling interrupts. Instead the driver can defer its own interrupts and pause execution of its own threads. This was sufficient in practice and has the advantage of lowering interrupt latency for other unrelated drivers.

Five drivers were ported for evaluating the system. The metrics used for the evaluation were complexity-measured by driver and specification size and performance-measured by throughput and latency and by robustness to random and targeted attacks. About 1% of the lines of code changed in each driver, and the specifications were between 100 and 150 lines of code—an order of magnitude smaller than the drivers themselves. The network throughput at 1 Gbps for user-space drivers degraded slightly for sending small packets, but no penalty was measured for sending large packets or receiving packets. The interrupt latency degraded significantly (from 5 to 50 microseconds), but that did not affect usability in day-to-day use. The CPU overhead for a benchmark streaming video at 1 Mbps increased from 1% to 2%. Finally, the resistance to attacks was evaluated by probabilistically modifying driver operations as well as using a malicious driver suite. Having the drivers in user space only caught direct reads/writes, but the security policies blocked all further attacks. Patrick concluded that the system is efficient, general, and robust against attacks.

A member of the audience objected to the use of throughput as a metric for the evaluation and highlighted that CPU load was roughly doubled. Another member wondered how this system would affect more demanding devices using Firewire or USB2.0. Patrick replied that latency and correctness were not affected for USB1.1 and USB 2.0 high-speed drives, although with USB 2.0 the CPU overhead is significant, as USB 2.0 drivers require many context switches. Patrick specifically clarified that they have not observed dropped frames with their USB disk experiments, but they have not tried Firewire. Somebody asked who would write the specifications. Patrick suggested that the device driver manufacturers or a trusted third party could write these, but it is an orthogonal problem. Asked whether performance was the reason for having the reference monitor inside the kernel, Patrick argued that it has to be part of the TCB, so it might as well be in the kernel, but he believes that the overhead of having it in user space would be small. Someone else observed that the specification can be extracted either from the driver's normal behavior or from the device's specification. Patrick said that they looked at both and considered it a shortcut to allow only observed behavior, but he agreed that for implementing a security policy normal behavior is more interesting than the device's full capabilities.

People were intrigued by the possibility of permanent damage to the hardware and solicited realistic examples. Patrick described a driver asking a device to overclock itself and turn off its fans or overwrite flash memory, rendering the device unbootable. Having observed that a generic USB policy was used for USB drivers, somebody asked for a comparison with the Windows driver model that also allows user-space drivers. Patrick acknowledged that they are very similar but argued that Windows allows only user-space devices that do not perform direct IO, whereas Nexus extends this to all kinds of devices. Asked whether an IOMMU would be useful for doing lazy trapping and batching things up, much as shadow page tables are used, Patrick observed that many memory accesses are individual commands that cannot be batched like that.

The final question was about similar description languages such as Devil IDL and Sing\#, the Singularity driver constraint language. Patrick said that they drew inspiration from the languages but did not reuse them or reuse any of the specifications that were written for them. He further argued that Devil was intended for the construction of drivers rather than constraining them, and helater added that Sing\# applies only to properties that can be checked statically.

- ***Digging for Data Structures***
  Anthony Cozzie, Frank Stratton, Hui Xue, and Samuel T. King, University of Illinois at Urbana-Champaign

Anthony Cozzie summarized the state of the art in the anti-virus industry and the challenges faced by signature checkers against code obfuscation. He argued that all programs use data structures and targeting these could defeat code obfuscation. To this end, he presented Laika, a system that can detect programs based on their data structures.

Laika works by classifying words in memory into types such as heap pointers, zero, integers, or ASCII strings, and from these it tries to find objects in memory and their classes. Importantly, in addition to field types, it can classify objects by considering what other objects their fields point to.

Anthony presented an evaluation of Laika's accuracy using seven test programs that averaged 4000 objects and 50 classes: Laika detected object classes with 75%–85% accuracy. It was straightforward to turn Laika into an anti-virus program by checking how many structures a tested program shares with a known malware sample. They tried this with three botnets, which Laika classified with 100% accuracy, whereas ClamAV, an anti-virus program, had about 80% accuracy. Anthony observed that virus detection is an arms race, and he discussed some possible attacks on the system and countermeasures. Obfuscating reads and writes would not help the virus, as it would result in no visible data structures, which is suspicious on its own. A compiler attack could shuffle the order of fields, but that would preserve sufficient information to recover the classes. A mimicry attack could use structures from a legitimate program, such as Firefox, but a successor to Laika could try to detect unused fields, or it could detect the mismatch between the program (not Firefox) and its data structures (Firefox). The approach does not work for very simple programs, but malware with some functionality would expose some structure. Finally, being a dynamic approach, it is computationally expensive.

Stefan Savage highlighted the challenges of getting a consistent memory snapshot, with malware being careful not to expose completely at once, or waiting until a particular time of day. Anthony said that they just run the malware for five minutes and that was sufficient, but he acknowledged the problem. Another attendee observed that the attacker code might try to detect whether it is in a sandbox or a VM, to which Anthony replied that the approach relies on a working sandbox. Another point raised involved the whole class of nonmalicious applications such as DRM that use obfuscation. Anthony argued that although they could not be classified by Laika, they could be signed, thus solving this problem.

Another attendee asked whether they have tried running Laika on programs written in other languages or using more exotic compilers. Anthony said that the results would depend on the language. For example, it would not work with LISP, where everything is a list, but with some other high-level languages such as Java, the classes are readily available. Somebody observed that many programs share data structures because of libraries and runtime environments, such as the C library and the JVM, and wondered how the technique would deal with such programs. Anthony argued that the approach can still separate such programs. Somebody proposed to break Laika by encrypting data structures on the heap and decrypting them on the stack before use, but Anthony classified this as another case of an encrypted memory attack where no structure would be visible and used a graphic analogy of cutting one's fingers to avoid matching a fingerprint scan, reiterating that lack of structure is suspicious by itself. Finally, somebody was concerned with having to run Laika for five minutes and wondered whether matching against several candidate viruses would take five minutes per virus signature, but Anthony clarified that one run of the scanned program and a snapshot are sufficient for checking multiple signatures.

DEALING WITH CONCURRENCY BUGS

Summarized by Chris Frost (frost@cs.ucla.edu)

- ***Finding and Reproducing Heisenbugs in Concurrent Programs***
  Madanlal Musuvathi, Shaz Qadeer, and Thomas Ball, Microsoft Research; Gerard Basler, ETH Zurich; Piramanayagam Arumuga Nainar, University of Wisconsin, Madison; Iulian Neamtiu, University of California, Riverside

Madanlal Musuvathi presented their work on making it easier to find concurrency bugs. Concurrent applications are infamously difficult to debug: Executions are highly nondeterministic, rare thread interleavings can trigger bugs, and slight program changes can radically change execution interleavings. For these reasons, concurrency bugs can cost significant developer time to find and reproduce. Madanlal et al.'s user-mode scheduler, CHESS, controls and explores nondeterminism to trigger concurrency bug magnitudes more quickly than through stress testing.

Madanlal showed the utility of CHESS through a well-received concurrency bug demo. He first ran a bank account application test suite hundreds of times; no execution triggered a concurrency bug. Importantly, over the hundreds of runs only two unique execution paths happened to be explored. He then ran the test suite in CHESS; the bug was triggered on the fourth run. Each run in CHESS explored a unique execution path.

The high-level goals of CHESS are to find only real errors and to allow all errors to be found. This latter suite goal is difficult because of the state explosion that results from all the sources of nondeterminism; practically, Madanlal et al. wanted CHESS to be able to beat stress checking.

CHESS is a user-mode scheduler; it explores execution paths in a test (1) to trigger test suite asserts and (2) for CHESS to detect deadlocks and livelocks. CHESS inserts scheduler calls at potential semantic preemption points. Several techniques are used to reduce the number of these scheduler calls to help minimize state explosion; some of these techniques can be adjusted to trade off speed with coverage. CHESS has found bugs in Microsoft's Concurrency Coordination Runtime, IE, and Windows Explorer, among others. It is available for Win32 and .NET now and will be available later for the NT kernel. They also plan to ship it as an add-on with Visual Studio. Finally, Madanlal made a plea for abstraction designers: Specify and minimize exposed nondeterminism.

One person asked whether CHESS controls random number generators and time. Madanlal responded that CHESS guarantees reproducibility for random number generation, timeouts, and the time of day, but it will not find errors that only show up on particular return values (i.e., it captures but does not explore the nondeterminism). CHESS leaves file and network inputs to the test suite.

- ***Gadara: Dynamic Deadlock Avoidance for Multithreaded Programs***
  Yin Wang, University of Michigan and Hewlett-Packard Laboratories; Terence Kelly, Hewlett-Packard Laboratories; Manjunath Kudlur, Stéphane Lafortune, and Scott Mahlke, University of Michigan

Yin Wang presented Gadara, a principled system that provably ensures that a multithreaded C program is deadlock-free, while requiring only a modest runtime overhead (e.g., 11% for OpenLDAP). Gadara avoids circular-mutex-wait deadlocks by postponing lock acquisitions, using a program binary instrumented with control logic automatically derived from the source program using discrete control theory. Gadara transforms a C program into a control flow graph, then into a Petri net (a discrete control theory model used to describe lock availability, lock operations, basic blocks, and threads), from which Gadara derives control logic that it adds to the original source. The control logic observes and controls thread interactions. Most lock operation sites can be ruled safe and thus need not be instrumented.

Yin et al. evaluated Gadara for the OpenLDAP server, BIND, and a PUBSUB benchmark. Gadara can require source function and lock annotations; the OpenLDAP server was mostly annotated in one hour (with 28 of 1,800 functions being ambiguous). Gadara found four possible sources of deadlock: two new, one known, and one false positive (resulting from Gadara not using data flow analysis). The known bug was fixed in the OpenLDAP repository, but it was later reintroduced, and then fixed again through a code redesign. With Gadara's runtime instrumentation no fix is necessary, avoiding this frequent deadlock bug fix difficulty. This makes Gadara useful for rapid development, corner cases in mature code, and end-user bug fixes.

One person asked why one would use Gadara's instrumentation instead of fixing the bug. Yin responded that developers are not always available, that fixes can be difficult to design, and that Gadara can generate false positives, necessitating code study to determine whether the bug is real. Another person asked why adding a simple lock around the instrumentation points is insufficient; this solution works in some cases, but not all (e.g., the five dining philosophers problem). In response to why the PUBSUB experiment used only two cores. Yin answered that they found, in the BIND experiments, that 2- and 16-core experiments had similar overheads. For both PUBSUB and BIND, only workloads that saturated the CPU showed any performance overhead.

- ***Deadlock Immunity: Enabling Systems to Defend Against Deadlocks***
  Horatiu Jula, Daniel Tralamazza, Cristian Zamfir, and George Candea, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

George Candea presented the Dimmunix deadlock avoidance system. When a system deadlocks, Dimmunix remembers the control flow that has brought the program into the current state and, through lock operation instrumentation, prevents the program from reentering that state in future program runs. George had fun describing their work in terms of antibodies (Dimmunix signatures) for pathogens (deadlock bugs): You get sick from the first infection, but you are immune from then on.

When a program deadlocks, Dimmunix saves the values for each frame in the deadlocked threads' call stacks. After the program is then restarted, if a lock operation is performed with the same callstacks Dimmunix changes the lock operation ordering to avoid the deadlock. Dimmunix is implemented for C/C++, as a modified POSIX thread library, and for Java, by rewriting Java bytecode. The talk and the paper detail the performance challenges they faced and overcame. Dimmunix avoids inducing thread starvation when reordering a program's lock operations using Dimmunix's deadlock avoidance algorithm with yield edges.

George sees Dimmunix as complementary to existing deadlock reduction techniques. For example, although a browser may have no deadlocks itself, a third-party plug-in may introduce deadlocks. An end user can use Dimmunix to avoid these, without source code access and without understanding the program's internals. George stated that Dimmunix has four important properties: (1) Someone must experience the first deadlock occurrence; (2) Dimmunix cannot affect deadlock-free programs; (3) Dimmunix cannot induce nontiming execution differences; and (4) Dimmunix must be aware of all synchronization mechanisms.

They evaluated Dimmunix by reproducing user-reported deadlock bugs for a number of large systems, including MySQL, SQLite, and Limewire, among others. For each reproducible deadlock they were able to induce deadlock, when not using Dimmunix, one hundred times in a row. With Dimmunix, each bug was triggered only the first of the hundred runs.

One person asked about extending Dimmunix to provide the stack traces to the programmer for debugging; George said one of his students is working on this for very large programs. Another asked what happens when Dimmunix misbehaves, for example, preventing a valid path from executing. George responded that Dimmunix protects against this using an upper bound on yielding. Another person asked whether Dimmunix could be generalized to segmentation faults or other symptoms; George said that another student is looking at applying their techniques to resource leaks.

POSTER SESSION

Part 1 summarized by Kyle Spaans (kspaans@student.math.uwaterloo.ca)

- ***Automatic Storage Management for Personal Devices with PodBase***
  Ansley Post, Rice University/MPI-SWS; Petr Kuznetsov, Juan Navarro, and Peter Druschel, MPI-SWS

Users don't like having to do work. PodBase is a system that tries to avoid involving the users in any way with their files. It is a system for synchronizing and replicating user data on personal devices, under the covers, for durability and availability. For example, it can use the freely available storage on the various devices in a user's household to back up files in case the original is lost.

- ***Entropy: A Consolidation Manager for Clusters***
  Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, and Gilles Muller, Ecole des Mines de Nantes; Julia Lawall, DIKU

Hermenier described Entropy, a system to optimize configuration in cluster environments for VMs. It packs and migrates VMs to reduce overhead and make best use of resources. For example, it can reduce the number of migrations needed, take advantage of parallelism, and minimize the number of nodes required.

- ***Writing Device Drivers Considered Harmful***
  Leonid Ryzhyk and Ihor Kuz, NICTA, University of New South Wales

Ryshyk said that device drivers are hard to write correctly and can compromise an entire system. This research aims to automate the process by taking a formal device driver specification along with the OS API and composing them into a driver program. The only effort necessary is to translate the device specification into a formalized finite-state machine. Ideally, this could be generated from the VHDL code of the device itself. To avoid state explosion in more complex drivers, symbolic interpretation is used.

- ***NOVA OS Virtualization Architecture***
  Udo Steinberg and Bernhard Kauer, TU Dresden

A very small microhypervisor that can run legacy OSes, NOVA, focuses on being the small trusted base (~30 KLOC) that runs in privileged mode on the hardware. It runs the maximum possible number of services in user space, monitors included, so that compromises cannot spread among VMs.

- ***CrystalBall: Predicting and Preventing Inconsistencies in Deployed Distributed Systems***
  Maysam Yabandeh, Nikola Knezevic, Dejan Kostic , and Viktor Kuncak, EPFL

CrystalBall makes it easier to find errors in distributed systems and increases their resilience. It will predict inconsistencies in live systems and then steer execution to avoid the predicted errors. It's a form of model checking and allows live distributed debugging with low overhead.

- ***Operating System Transactions***
  Donald E. Porter, Indrajit Roy, and Emmett Witchel, The University of Texas at Austin

Secure sandboxing is hard. Sandboxed systems can still be vulnerable to timing attacks. OST's approach is to atomicize and isolate access to system calls using a simple API to give more transactional behavior. It is implemented with lazy version management and eager conflict resolution.

- ***The Network Early Warning System: Crowd Sourcing Network Anomaly Detection***
  David Choffnes and Fabian E. Bustamante, Northwestern University

NEWS uses distributed clients to detect anomalies in network performance and can be useful as another tool in a network administrator's toolbox. As usual, it is a tradeoff between coverage and overhead, but network overhead is kept minimal by taking advantage of the fact that most P2P applications already implicitly monitor the network anyway. NEWS is implemented as a Vuze (BitTorrent client) plug-in, with a central reporting Web interface useful for administrators.

Part 2 summarized by Kathryn McBride (katymcbride@yahoo.com)

- ***File System Virtual Appliances: Third-party File System Implementations without the Pain***
  Michael Abd-El-Malek, Matthew Wachs, James Cipar, Elie Krevat, and Gregory R. Ganger, Carnegie Mellon University; Garth A. Gibson, Carnegie Mellon University/Panasas, Inc.; Michael K. Reiterz, University of North Carolina at Chapel Hill

No summary available.

- ***The Barrelfish Multikernel Operating System***
  Andrew Baumanny, Simon Peter, Jan Rellermeyer, Adrian Schüpbach, Akhilesh Singhania, and Timothy Roscoe, ETH Zurich; Paul Barham and Rebecca Isaacs, Microsoft Research, Cambridge

Barrelfish is a new operating system that is being built from scratch. Barrelfish uses a rapidly growing number of cores, which leads to a scalability challenge. Barrelfish gives the user the ability to manage and exploit heterogeneous hardware and run a dynamic set of general-purpose applications all at the same time. The researchers at ETH Zurich and Microsoft Research, Cambridge, are exploring how to structure an operating system for future multi- and many-core systems with Barrelfish.

- ***Dumbo: Realistically Simulating MapReduce for Performance Analysis***
  Guanying Wang and Ali R. Butt, Virginia Tech; Prashant Pandey and Karan Gupta, IBM Almaden Research

Dumbo provides a good model for capturing complex MapReduce interactions and predicts the performance of test clusters. Dumbo aids in designing emerging clusters for supporting MapReduce. Dumbo takes metadata, job descriptions, and cluster topology to the ns-2 server, where it is traced.

- ***Aggressive Server Consolidation through Pageable Virtual Machines***
  Anton Burtsev, Mike Hibler, and Jay Lepreau, University of Utah

No summary available.

- ***Scalable Fault Tolerance through Byzantine Locking***
  James Hendricks and Gregory R. Ganger, Carnegie Mellon University; Michael K. Reiter, University of North Carolina at Chapel Hill

No summary available.

- ***Eyo: An Application-Oriented Personal Data Synchronizer***
  Jacob Strauss and Chris Lesniewski-Laas, MIT; Bryan Ford, MPI-SWS; Robert Morris and Frans Kaashoek, MIT

Eyo was developed out of the interference challenges on handheld devices. All types of media are produced and consumed everywhere on laptops, phones, and MP3 players. Eyo is used to transfer files seamlessly to the right place at all times. Central servers can limit usability. There is currently no offline client synchronization. Partial replicas have vast differences in storage capacity (e.g., a phone versus a laptop). There are also bandwidth limitations. Eyo is an attempt to synchronize personal data using an application-orientated approach.

- ***ProtoGENI: A Network for Next-Generation Internet Research***
  Robert Ricci, Jay Lepreau, Leigh Stoller, Mike Hibler, and David Johnson, University of Utah

No summary available.

Part 3 summarized by John McCullough (jmccullo@cs.ucsd.edu)

- ***NetQuery: A Universal Channel for Reasoning about Network Properties***
  Alan Shieh, Oliver Kennedy, and Emin Gün Sirer, Cornell University

Information about networks and their endpoints is scarce. If a service wanted to restrict access to protected networks or if a client wanted to connect to an ISP with better provisioning or DDoS protection, they would be hard pressed to find the information on their own. NetQuery uses tuplespaces to store such useful information. The tuplespace abstraction provides for filtering and modification triggers, enabling the rapid dissemination of reputable network information.

- ***Trapper Keeper: Using Virtualization to Add Type-Awareness to File Systems***
  Daniel Peek and Jason Flinn, University of Michigan

Collecting file metadata requires intimate knowledge of the file formats. It is straightforward to read metadata from common file formats with well-known parsers. However, there are thousands of obscure file formats that don't have readily accessible parsers; it is impractical to program parsers for all of them. Trapper Keeper leverages the parsers in applications by loading the files and extracting the information from the accessibility metadata in the GUI. Manipulating a GUI for each individual file in the file system is problematic. However, using virtualization, the application can be trapped and snapshotted when it is about to open a file. Thus, the snapshot can be invoked with a file of interest. Trapper Keeper can then use this technique to extract metadata from all of the files in your file system.

- ***Gridmix: A Tool for Hadoop Performance Benchmarking***
  Runping Qi, Owen O'Malley, Chris Douglas, Eric Baldeschwieler, Mac Yang, and Arun C. Murthy, Yahoo! Inc.

As Hadoop is developed, it is important to have a set of applications that exercise the code base. Gridmix is an open set of applications useful for benchmarking, performance engineering, regression testing, cluster validation, and configuration evaluation. The current application set has been critical in the recent performance enhancements in Hadoop. Gridmix is publicly available in the Hadoop source repository.

- ***Performance Evaluation of an Updatable Authenticated Dictionary for P2P***
  Arthur Walton and Eric Freudenthal, University of Texas at El Paso

An authenticated dictionary provides key-value pairs that are certified by an authority. Such a dictionary could be used to maintain blacklists for DHT membership on end-user machines. Fern is a scalable dictionary built on Chard that uses a tree to hierarchically distribute the potentially rapidly changing data such that the data is cacheable. The values can be validated by tracing a path to the root authority. Because the validation step has latency proportional to the height of the tree, it is desirable to keep the tree as short as possible. This work explores how the branching factor of the tree can reduce the height and how per-node load is affected as a result.

- ***Miser: A Workload Decomposition Based Disk Scheduler***
  Lanyue Lu and Peter Varman, Rice University

Ensuring high-level quality of service for all disk requests necessitates a significant degree of overprovisioning. However, the portion of the requests that necessitate this overprovisioning can be less than 1% of a financial transaction workload. Relaxing the low-latency QoS requirements for a fraction of the requests greatly reduces the provisioning requirements. The differentiated service is provided by multiple queues. The priority requests are serviced from one queue and the other requests are serviced in a best-effort manner using the slack of the priority queue.

VARIOUS GOOD THINGS

Summarized by Ann Kilzer (akilzer@gmail.com)

- ***Difference Engine: Harnessing Memory Redundancy in Virtual Machines***
  Diwaker Gupta, University of California, San Diego; Sangmin Lee, University of Texas at Austin; Michael Vrable, Stefan Savage, Alex C. Snoeren, George Varghese, Geoffrey M. Voelker, and Amin Vahdat, University of California, San Diego

  Winner of Jay Lepreau Award for one Best Paper

Amin Vahdat presented Difference Engine, which was awarded Jay Lepreau Best Paper. The motivation for the work is the increasing trend toward server virtualization in the data center to support consolidation and, ultimately, cost reduction. Some hurdles to consolidation include memory limits on virtual machines and bursty CPU utilization. The work is based on two concepts: first, memory-saving opportunities extend beyond full-page sharing and, second, page faults to memory cost less than page faults to disk.

Contributions of the Difference Engine include comprehensive memory management for Xen, efficient memory management policies, and detailed performance evaluation. Difference Engine finds opportunities for memory savings through identical page sharing, page patching, and compression of pages unlikely to be accessed in the near future. Three challenges here are (1) choosing appropriate pages for sharing, patching, and compression, (2) identifying similar pages, and (3) memory overcommitment. To combat the first challenge, the system uses a global clock to see which pages have been recently modified or accessed. For the second, Difference Engine operates by keeping two hash tables, one for sharing and one for similarity. On identical hashes into the sharing table, Difference Engine verifies byte-by-byte equality before enabling page sharing. To identify similar pages with low overhead, the system hashes over subpage chunks. To deal with memory overcommitment, the system implements demand paging in the VMM.

The Difference Engine is built on top of Xen 3.0.4, and the source code is publicly available at http://resolute.ucsd.edu/hg/difference-engine. For evaluation, the authors used micro-benchmarks of the cost of individual operations. They measure memory savings and performance over 10-minute intervals after stabilization on homogenous and heterogeneous workloads. The results show a memory savings of up to 90% in homogenous workloads, gained primarily from page sharing. Heterogeneous workloads saw a memory savings of up to 65%. Performance overhead was less than 7% compared to the baseline with no Difference Engine.

One audience member inquired whether the complexity of compression outweighed the 5% benefit. Vahdat noted that compression was run after page sharing and patching and has limited complexity. Switching the order of memory saving techniques could contribute greater savings to compression. Another audience member asked about the size of the reserve of free pages kept to resolve page faults. Although Difference Engine's current implementation uses a fixed number, Vahdat noted that it would be straightforward to make this value a percentage. There was a follow-up question regarding server node paging. Vahdat explained that some amount of paging occurred during startup, but there was very little paging during the performance evaluations. One questioner asked about the benefits of only using demand paging in the hypervisor and identical page sharing, to which Vahdat answered that memory could be reduced by an additional factor of 1.6 to 2.6 when additionally employing patching and compression for heterogeneous workloads.

[See p. 24 for an article on Difference Engine by the authors of this paper.—Ed.]

- ***Quanto: Tracking Energy in Networked Embedded Systems***
  Rodrigo Fonseca, University of California, Berkeley, and Yahoo! Research; Prabal Dutta, University of California, Berkeley; Philip Levis, Stanford University; Ion Stoica, University of California, Berkeley

Prabal Dutta presented Quanto, a system for tracking energy in sensor networks, where it is the defining constraint. Sensor nodes ("motes") spend most of their time sleeping and often run on limited energy supplies, such as AA batteries. These nodes consume energy in very short bursts and they display orders of magnitude difference in current draw between active and sleep states. Three basic challenges in energy tracking include metering energy usage, breaking down energy usage by device, and tracking causally connected activities both within a node and across the network.

To measure energy usage, Quanto uses the iCount energy meter. To break down energy usage, device drivers are instrumented to expose power states, and Quanto uses this information, along with knowledge of the aggregate system energy usage, to estimate power breakdowns by device using regression. For activity tracking, Quanto labels executions, and these labels are propagated throughout the system and its device drivers, and even onto other motes. The labels help identify the origin and execution path as belonging to a particular activity.

Quanto is built on top of TinyOS, and it only needs 12 bytes per activity or energy sample. In their RAM-based logger, Quanto keeps a buffer of 800 samples. The authors evaluate Quanto regression by testing 48 seconds of the Blink application and comparing the results with ground truth obtained using an oscilloscope. The results show that, although Quanto takes a large portion of active CPU time (71%), the denominator is very small, since actual CPU active time is 0.12%, and most of the time the sensor is sleeping. Additionally, Quanto is able to monitor itself.

Quanto is very useful for measuring energy use and CPU time for sensor network applications. Dutta provided a monitoring example where Quanto was used to evaluate the cost of false alarms in low-power listening. Additionally, the application Bounce, which plays ping-pong with network packets, demonstrates Quanto's ability to track an activity propagating from node to node. Some ideas for future work include scaling Quanto from initial tests with just two nodes to a large-scale testbed of 1000 nodes, examining the energy cost of various network protocols, and developing systems for energy management.

After the presentation, an audience member observed that Blink and Bounce were simple applications and asked how Quanto scaled to larger programs. Dutta noted that the full profile logging generated quite a bit of data, even for these small programs, and complex problems were difficult to measure in the current version of Quanto, but that these issues had been addressed in subsequent work. Another person asked about analyzing energy online rather than offline. Dutta suggested keeping counters instead of profile data and calculating periodic regressions to allow online usage of Quanto. Noting the challenge of getting device drivers to model device state, one audience member asked Dutta what sorts of hardware modifications would be useful to directly extract state information. Dutta replied that it would be great to get power state information directly from the hardware, but this might be infeasible in some complex subsystems (e.g., hard disks).

- ***Leveraging Legacy Code to Deploy Desktop Applications on the Web***
  John R. Douceur, Jeremy Elson, Jon Howell, and Jacob R. Lorch, Microsoft Research

Jon Howell presented Xax, a system for running legacy code securely within Web browsers. He began by observing the popularity of Web 2.0 applications, which allow location independence, OS independence, and isolation to browser windows. Howell also observed that decades of knowledge and hard work remained in legacy C/C++ code. It would be nice to still use this code, but rewriting it or maintaining multiple code bases is often a significant obstacle. Running the code on a remote server is not a viable solution, because it would take too long. Running the code in a client is not OS independent and presents security concerns.

Xax solves the problem of running legacy code on a client by creating a Xax container. The container, called the "picoprocess," is an OS process, which provides MMU isolation, and the system call interface is filtered through a Xax monitor. Services are provided through the browser. Additionally, Xax has a platform abstraction layer to allow OS independence. It requires light code modification. Benefits of Xax include native CPU performance, legacy support, and security via isolation.

To show that Xax will work, the authors built many demos. Howell demonstrated a clock, an openGL example, and an implementation of Ghostscript. The "lightweight code modification" involved changes such as removing static dependencies, rejecting unnecessary system calls, and making I/O operate through the browser. The authors evaluated Xax by using it to support 21 libraries and 3.7 million lines of code. In conclusion, Howell summarized Xax as a secure, fast, and portable interface for running legacy code in a Web browser.

The first question regarded limitations on Xax. Howell said that the openGL example was limited because it relied on compressed PNGs for display. If Xax could get a low-level interface to the browser, it could contain its own rendering engine. Howell noted that Xax just needed the right low-level interface to provide greater functionality. Another audience member asked about the 3.7 million lines of code, and how much of it was actually being run by Xax. Howell noted that most of this code was being used, and that "shims" (parts specific to the X server or OS) were cut out.

One person asked about security and limiting resource usage. Howell noted that resource constraints would be easy to add to Xax. Another attendee asked about performance and the amount of data transferred over the Web. He also inquired whether dial-up users would be able to run Xax. Howell noted that they didn't worry about these issues when building the first version of Xax, because Web caching and other software engineering tools could be added later to make Xax more efficient.

[See p. 32 for an article on Xax by the authors of this paper.]

WIDE-AREA DISTRIBUTED SYSTEMS

Summarized by Roxana Geambasu (roxana@cs.washington.edu)

- ***FlightPath: Obedience vs. Choice in Cooperative Services***
  Harry C. Li and Allen Clement, University of Texas at Austin; Mirco Marchetti, University of Modena and Reggio Emilia; Manos Kapritsos, Luke Robison, Lorenzo Alvisi, and Mike Dahlin, University of Texas at Austin

Harry Li presented FlightPath, a peer-to-peer system for media streaming applications that is able to maintain low jitter in spite of Byzantine or selfish peers. This work is motivated by the observation that most of today's cooperative systems lack rigorously defined incentives, which leaves room for exploits and free-riders. The author made reference to their previous work, which used Nash equilibria from gaming theory to provide provable incentives for rational users not to deviate from the protocol. That work, as well as other related works, however, sacrificed flexibility and performance for correctness. In this work, Harry and his co-authors aimed at approximating Nash equilibria to achieve both formal incentives and efficiency.

More specifically, they propose an epsilon-Nash equilibrium scheme, in which rational peers may only gain a limited advantage (< an epsilon) from deviating from the protocol. This provides nodes with some freedom in choosing peers, which in turn allows them to steer away from overloaded peers and avoid departed peers. The author stressed that it is this flexibility that enables some of the properties of FlightPath: churn resilience, byzantine and rational peer tolerance, and high-quality streaming.

A member of the audience asked about FlightPath's resilience to collusion attacks. The author answered that they had considered collusion for all of the results reported for malicious attacks. Another member of the audience pointed out that a previous study had shown that most free-riders accounted for little bandwidth in a collaborative system. He wondered to what degree eliminating those small-resource free-riders would improve overall performance in a real collaborative system.

- ***Mencius: Building Efficient Replicated State Machines for WANs***
  Yanhua Mao, University of California, San Diego; Flavio P. Junqueira, Yahoo! Research Barcelona; Keith Marzullo, University of California, San Diego

Yanhua Mao presented a Paxos-based replication protocol specifically designed for WAN operation. The author envisions this protocol to be useful in cross-datacenter geographical replication. The author explained why current Paxos protocols (Paxos and Fast Paxos) perform poorly on WANs. On one hand, Paxos maintains a single leader and thus achieves poor latency for operations issued at nonleader replicas. On the other hand, Fast Paxos achieves good latency by allowing all replicas to behave as leaders, but it suffers from collisions, which lead to poor throughput.

The proposed system, Mencius, aims to take the best of both worlds. Very briefly, their approach consists of two mechanisms: rotating leader and simple consensus. The former allows the leader function to be assumed by the servers in a round-robin fashion, which means equal latencies and high throughput. The latter mechanism allows servers with low client load to skip their turn in Paxos rounds efficiently.

The authors evaluated the system by comparing Mencius's throughput and latency against Paxos's. A member of the audience asked the speaker to clarify a discrepancy in one of the graphs, which showed Mencius's throughput degrade gracefully after a crash, whereas Paxos's throughput was at an all-time low value before and after recovery. The author responded that this effect was due to Mencius's ability to use all of the servers' bandwidth, whereas Paxos was bottlenecked by the single leader's bandwidth.