Conference Reports

In this issue:

9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)

Vancouver, BC, Canada
October 4–6, 2010

Kernels: Past, Present, and Future

Summarized by Brendan Cully (brendan@cs.ubc.ca)

An Analysis of Linux Scalability to Many Cores
Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich, MIT CSAIL

At the previous OSDI, Silas Boyd-Wickizer presented Corey, a new many-core operating system motivated by the premise that traditional OS design, as represented by Linux, faced fundamental limitations to scalability. This talk, also given by Boyd-Wickizer, refutes that premise. It does so by selecting several real applications with good parallel implementations that demonstrate kernel-based scalability problems and exploring the root causes of the bottlenecks.

Boyd-Wickizer et al. did their examination on an off-the-shelf 48-core Linux x86 server, running the applications on a RAM disk to avoid disk bottlenecks and stress the kernel. They found the bottlenecks, fixed them, and kept repeating this until either scalability was linear or the bottleneck was not in the kernel (i.e., it was in hardware or the application itself). They were able to fix most of the kernel bottlenecks with 16 patches and 3000 lines of code, demonstrating that Linux can scale well up to at least 48 cores. Most of the remainder of the talk consisted of walkthroughs of the specific bottlenecks found and the fixes applied. All of the fixes were straightforward applications of classic parallel techniques, typically to reduce cache coherence and memory bus contention by manually maintaining cache-local copies of shared data structures.

Eric Van Hensbergen from IBM Research asked whether the authors were attempting to integrate their changes into upstream Linux. Boyd-Wickizer responded that they hadn't even attempted it, because the engineering effort required

was better spent on other research. Most other questions concerned the tension between this paper and the one on Corey from the previous year. Margo Seltzer (Harvard) and Eddie Kohler (UCLA) both asked variants of the question, "If shared state is bad for scalability, is it better to remove it at the points where it causes problems, or to avoid it generally, using it only at points where it is known not to cause problems?" Hank Levy (UW) put it most succinctly, getting a big laugh, when he asked, "Should we be designing new OS architectures or not? In other words, could you have written these papers in the opposite order?" Boyd-Wickizer carefully replied that Linux did not appear to have scalability problems up to 48 cores, but the possibility existed that it might after that.

Trust and Protection in the Illinois Browser OS

Shuo Tang, Haohui Mai, and Samuel T. King, University of Illinois at Urbana-Champaign

Shuo Tang presented the Illinois Browser OS (IBOS), an OS built from scratch to run Web browsers. It was motivated by two ideas: first, that vulnerabilities become more serious as they get deeper into the application stack, from individual Web application into the browser and finally the OS; and second, that it is easier to secure a system if the security layer has more semantic knowledge of what it is protecting.

Previous efforts like Google Chrome have used OS process isolation to provide some protection between compromised Web applications and the rest of the system, but IBOS takes the next logical step: it runs each browser in its own virtual machine with hardware-enforced isolation from the rest of a user's system. Its system call interfaces map to browser semantics much better than an ordinary operating system's would. For instance, instead of providing disk and network interfaces, it offers system calls for fetching URLs and storing cookies. This allows security policy to specify more precise invariants. By customizing the set of OS services to a Web browser, IBOS is also able to dramatically reduce the total TCB compared to an ordinary browser running on top of Linux. Tang concluded the presentation by revealing that his slides were served by a Web browser running on top of IBOS.

Shriram Rajagopalan of UBC asked if IBOS allowed the full set of services provided by modern browsers. Tang stated that IBOS used WebKit and supported most current browser features, including HTML 5. Etienne Le Sueur (NICTA) asked whether IBOS prevented cross-site scripting attacks and, if so, whether that broke beneficial uses such as Google Web Toolkit applications. Tang maintained that IBOS provided an enforcement mechanism, but using it was dependent on the user's security policy. The most fundamental question was from James Mickens (MSR), who noted that more and more Web browser research seemed to be retasking older core OS mechanisms, and wondered whether Tang believed that there was anything fundamentally different about Web browsers that made this reuse interesting. Tang simply replied that since Web browsers were running more general-purpose applications, we should be borrowing more operating system techniques.

FlexSC: Flexible System Call Scheduling with Exception-Less System Calls

Livio Soares and Michael Stumm, University of Toronto

System calls are expensive, both because of the time required to switch into kernel mode and because of cache contention while bouncing back and forth between user and kernel code. The traditional interface to the kernel is synchronous system calls, which make it impossible to amortize these costs. Livio Soares proposed an alternative kernel interface, FlexSC, which makes system calls asynchronous. This allows calls to be batched and, as a side effect, makes it possible to run kernel code on different cores from the user code it services. He also proposed a wrapper for FlexSC, called FlexSC-Threads, that hides the new system call style behind a traditional synchronous, thread-based API to support legacy code without major changes.

Soares demonstrated the degree to which system calls can reduce application performance, by modifying an application that makes very few system calls (Xalan from the SpecCPU benchmark) so that it made a large number of null system calls. In theory, the time excluding kernel processing to complete the benchmark should be about the same, but in fact the null system calls could halve performance, due to cache and TLB evictions. His solution was to use shared memory to queue system call requests. This not only allows batching of system calls, but can avoid context switches entirely if a separate core is dedicated to kernel request processing. Comparisons between MySQL and Apache using FlexSC showed up to 115% improvement in the Apache benchmark on four cores.

Van Hensbergen again asked whether Soares would attempt to integrate his interface into Linux. He received the same response given by Boyd-Wickizer at the first talk: it would be a lot of work and Soares would prefer to spend the energy on other things. Many others asked detailed questions about overhead measurement. For example, Michael Vrable wondered if the authors had measured cache contention due to userspace rescheduling when threads blocked. Soares had not, and he believes that this could indeed be another source of lost performance. Sasha Federova (SFU) noted that for FlexSC-Threads to show benefits, you would require more

threads than cores; she wondered whether this mode might cause degradation for some workloads. Soares agreed that the benefits would mostly be for server workloads and that scientific applications that were 100% CPU-bound wouldn't need this.

Inside the Data Center, 1

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

Finding a Needle in Haystack: Facebook's Photo Storage

Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel, Facebook Inc.

Peter Vajgel explained that Haystack is a photo storage system that addresses Facebook's significant scalability needs. Since April 2009, Facebook's photo storage has grown from 15 billion to 64 billion images. The update rate has increased similarly, from 220 million uploads per week to one billion currently. Prior to Haystack, pictures were served from a Content Distribution Network (CDN) that caches photos from an NFS server. Eventually, Facebook's working set grew to become so large that the cache hit rate through the CDN's had declined to 80% and the system bottlenecked on metadata access. At this larger scale, each photo access required 10 IOPS, due to deep directory lookups and file fragmentation. Through caching and reducing the directory sizes, they reduced this overhead to 2.5 IOPS. Haystack was developed to further lower the overheads of accessing a file to a single I/O operation, while maintaining a simple system based on commodity hardware.

Logically, the system is built from Haystacks, which are append-only object stores, and Needles, which are the photos and their associated metadata. Haystacks are generally 100GB in size and are stored on 10TB XFS volumes in a RAID6 configuration. The Haystack photo server builds a compact index of these images. The indexes for 10TB of images can be stored in 5GB, making the index less than 2% of the size of inodes. The system also supports load balancing and caching, and it operates over HTTP. In closing, Vajgel reiterated the goal of serving each request in a single operation and provided some hints at future work. The team plans to consider other RAID configurations and to investigate the use of flash-based storage.

Jason Flinn from the University of Michigan asked if prefetching was possible. Vajgel replied that users often select individual pictures from a page of thumbnails organized by some tag. This looks entirely random to Haystack. Christopher Colohan from Google asked if social networking could provide some clues to what photos would be accessed. Vajgel said that this might be possible, but that it would be best done at the application level, well above Haystack. Vipul Mathur from Network Appliance asked if migrating between Haystacks would be useful for load balancing, but Vajgel thought that RAID6's poor support for mixed workloads would limit such an approach, and he noted that the existing load balancer seemed sufficient.

Availability in Globally Distributed Storage Systems

Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan, Google, Inc.

Murray Stokely began by posing questions of data availability, the causes of unavailability, and methods for tuning to the observed environment. To find answers, the authors gathered a year's data from tens of Google's clusters. They found that most unavailability periods are transient with a median length of 15 minutes. However, unavailability durations differ significantly by cause. The median time to restart a node's storage software is 1 minute, but it is 10 minutes for a planned reboot, and a few minutes longer for an unplanned reboot. Interestingly, 37% of failures observed were correlated in time to other failures. Larger-scale outages of this type regularly share common causes: for example, software bugs, shared hardware components, or supply chain issues.

Florentina Popovici then explained how these results informed the development of analytical models to study the availability of large distributed storage systems. This approach led to several insights. For small to medium bursts of failures and large encodings, when data is striped across many nodes, rack-aware placement can increase the MTTF by a factor of 3. Using a Markov model for analysis, another insight was that improving data availability below the node layer did little to help, although reducing node failure rates has a significant effect on availability. She concluded by stressing that a highly available distributed storage system requires exploring beyond disk failure rates, and that correlated failures are important in understanding large-scale storage availability.

Garth Gibson from CMU commented on graphs that appeared to capture unavailability due to rolling updates. He saw potential in making active decisions about how these events occur. Stokely agreed, pointing to the possibility of making additional replicas before a planned reboot. Mehul Shah from HP Labs asked if Reed-Solomon encoding doesn't also offer cost savings and asked why that wasn't used initially. Popovici explained that there may be performance impacts, such as parallelism in batch-oriented processing, recovery bandwidth concerns, and the need for more servers and CPUs. Philip Howard from PSU asked how expected increases in CPU core density would affect correlated failure

rates. Stokely confirmed that this is a motivating question for the work and that the cost per core per watt per year is the determining factor. He also noted that the same questions can be analyzed about the number of disks per server and other parameters with this model.

Nectar: Automatic Management of Data and Computation in Datacenters

Pradeep Kumar Gunda, Lenin Ravindranath, Chandramohan A. Thekkath, Yuan Yu, and Li Zhuang, Microsoft Research Silicon Valley

Li Zhuang observed that a large fraction (20–40%) of computation in their datacenters is redundant and that most storage is occupied by datasets that are never accessed. The two inefficiencies share a common problem—intermediate data created as part of larger processing operations. Nectar attempts to increase datacenter efficiency by tracking and reusing these intermediate results, while also allowing them to be deleted safely when the need arises.

Nectar is designed for datacenter applications written in the dataflow language LINQ, which may involve multiple stages of computation being stored persistently. As these intermediate files are created, Nectar records the task that created them and its associated parameters. Since the environment is deterministic, future attempts to run the same task are replaced with much more efficient accesses of the cached data. Under storage pressure, a mark and sweep garbage collector identifies intermediate files that can be deleted and regenerated on demand. Since usage information is tracked, datasets that are old and infrequently accessed can be prioritized for deletion. Zhuang concluded by describing Nectar as a promising approach that automates and unifies the management of data and computation in the datacenter.

Zhuang fielded several questions about Nectar's design and the DryadLINQ environment. In addition, David Schultz of MIT observed that storage capacity sizes relative to throughput seem to suggest that memory and I/O bandwidth are more important than capacity. Zhuang stressed that over time, it's important that capacity and computation aren't wasted as well. Micah Brodsky of MIT asked if the higher-level semantics and policies we have been adding to these dataflow environments are leading us towards reinvention of distributed databases, or if we are inventing something new. Zhuang replied that while they learned from distributed databases, there are significant differences that influence design. Cited examples included the scale of operation and the design of materialized views.

Security Technologies

Summarized by Nathan Taylor (tnathan@cs.ubc.ca)

Intrusion Recovery Using Selective Re-execution

Taesoo Kim, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek, MIT CSAIL

Software bugs, misconfigured security policies, and weak passwords all routinely lead to compromised systems. When a break-in is discovered, system administrators are caught between simply rolling back to the latest backup and losing users' recent work, and spending an inordinate amount of time on a manual recovery, with the chance of missing a change caused by the adversary perpetually looming. To this end, the authors present a tool for system recovery, dubbed Retro, that can tease apart exploit-related changes from benign changes made by legitimate users.

Zeldovich began by contrasting their work to two straw-men arguments: the first involved a naive taint tracking scheme, where all tainted files are restored from an earlier backup. This exhibits the usual explosion problem common to taint tracking. The second, in-VM replay, is too slow, and external state changes, such as crypto keys, mean that benign replay may go off the rails. Retro's approach involves selective execution, wherein execution data such as function calls and syscall arguments are logged in a so-called action history graph. This graph is used to roll time back to the compromise of the system and replay all non-attack activity, while skipping replay of operations the system has concluded were not affected by the attacker. If the attack involved communication with the outside world, this activity cannot be resolved by the system; the recovery will pause and prompt the sysadmin for guidance.

Ten real-world and synthetic challenge attacks were used to evaluate the tool; six were able to roll back automatically, while the remaining four needed some form of user input to handle legitimate network traffic or skip over the attacker's SSH login attempts. The amount of time needed to restore a system was found to be a function of the number of objects to track, and not the length of the log. The runtime overhead varied depending on workload: a HotCRP server averaged 4GB/day with a 35% slowdown, while a continuous kernel compilation averaged 150GB/day with a 127% slowdown. However, pushing the logging to a spare core significantly improved performance.

The presentation drew a lively Q&A session. Josh Triplett of Portland State asked about recording fewer user actions to pay a lower CPU/storage overhead. Zeldovich replied that this is possible, at the expense of reconstructing more of the dependency graph at repair time. Margo Seltzer of Harvard

noted the similarities to providence and asked about what constitutes "acceptable" nondeterministic execution at repair time. Zeldovich argued that anything that the execution could have originally created was fair game. An audience member wondered about trying to explode the repair space by creating many false positives, to which Zeldovich pointed out that activity that taints everything could itself be a symptom of an attack. Lorenzo Cavallaro (Vrije Universiteit) asked about porting to Windows. Zeldovich replied that the ideas were applicable to Windows, but several more managers would have to be written. They use SELinux to protect Retro, and something would be needed to replace that as well.

Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications

Adam Chlipala, Impredicative LLC

Enforcing sensible security policies is a challenge if a vulnerability relies on semantic knowledge of the application, such as a particularly crafted URL or a magic value within a cookie. Developers need to find attack vectors and audit them, but one can always be blindsided by a vector that one had not considered. Chlipala argued that it's far better to isolate a resource and have a policy stating how said resource may be accessed. He went on to present UrFlow, a SQL-like policy language that exploits a common database query–like idiom to define what parts of a program have access to a sensitive resource. Chlipala argued that UrFlow has the best of both static and dynamic checking: no source-level annotations are required, and all codepaths can be checked without incurring a runtime overhead.

UrFlow translates its policies into simple first-order logic for modeling what the system should know to be true when matching against policies; for instance, after reading a cookie, UrFlow knows that the system will know a password. Program paths are checked against its policies statically. Each execution path is executed symbolically, and as each progresses, the new logical clauses it has generated are sent to the theorem prover.

UrFlow was tested against applications ranging from ~100 to ~500 lines of code with ~50 policies each. Static checks are usually performed in well under a second. There were some significant outliers in runtime, but Chlipala hoped that applying traditional optimization tricks would smooth that out. Chlipala finished his talk by highlighting a disconnect of PLT researchers and programming practitioners: it's hard to sway developers away from imperative languages, but most Webapps use SQL, a perfectly reasonable declarative language anyway!

Most of the Q&A session was concerned with writing policies in practice. James Pendergrass (APL) asked if there is a way to ensure that rules are consistent with each other; Chlipala admitted that this is a tricky problem. Along the same lines, Pendergrass asked whether UrFlow is amenable to growing policies over time. Chlipala replied that it would be, since you won't be able to accidentally undo any previous policy. Joshua Triplett asked whether the theorem prover could be improved to provide hints about how the code could be changed to guarantee security. Chlipala admitted that it's tough to determine which fact in the rule base caused the prover to error out, but that there might be ways to heuristically prune it.

Accountable Virtual Machines

Andreas Haeberlen, University of Pennsylvania; Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel, Max Planck Institute for Software Systems (MPI-SWS)

Haeberlen presented a running scenario of a multiplayer game where players would like to guarantee that no player is using a cheat, and in this way they addressed the more serious problem of knowing whether programs on a third-party machine are executing as intended. A user should be able to detect when a remote machine is "faulty" and provide evidence of the fault to a third party without knowing the internals of their software. The authors' solution to this uses so-called accountable virtual machines. AVMs use a log of network traffic, signed with other players' keys, as a secure certificate of correct execution. Players may replay each others' logs in their own VM. If a player is running the game with a cheat, such as the "infinite ammo" hack, control flow will diverge from the log when the unhacked reference VM's game runs out of bullets. In so doing, strong accountability for arbitrary binaries is offered, without needing to trust other players or the AVM itself.

MPI-SWS built a prototype AVM atop VMware Workstation and tested it on Counterstrike 1.6. Performance is reasonable; on native hardware, the game averages 158FPS, and the authors observed a 13% slowdown with the AVM, mainly due to the overhead of the execution logging. The log size grows at 8MB/min, but because each action is replayed, replaying takes about as long as the game itself. A sample of 26 real-world CS cheats were all detected. But Haeberlen wonders whether cheaters could adapt their cheats. Re-engineering hacks to evade AVM detection, cheaters would have to fill in missing interrupts with the right branch counter value, which, although very hard, is theoretically possible.

Neal Walfield (Johns Hopkins) wondered whether the AVM itself could be hacked to fake the log. Haeberlen responded that the authenticators sign the AVMs and invited Walfield to discuss this offline. The subsequent questions involved

seeking clarification on what class of cheats the AVM can detect. Haeberlen reiterated that the AVM is guaranteed to find the attack unless the cheat is "plausible" input; for instance, a better-aim cheat might not be detected. On the other hand, because there's no input that could drive the program into such a state, the "infinite ammo" cheat is detectable. Brad Chen was skeptical of the applicability of the AVM to domains outside gaming, since the act of replaying requires sharing a great deal of information. Haeberlen admitted that the auditor has to be able to see everything, but he pointed out that in a cloud computing scenario a cloud owner ought to be able to audit his own VM.

Concurrency Bugs

Summarized by Kaushik Veeraraghavan (kaushikv@umich.edu)

Bypassing Races in Live Applications with Execution Filters

Jingyue Wu, Heming Cui, and Junfeng Yang, Columbia University

Jingyue Wu described a use-after-free data race in Mozilla's JavaScript engine with a simple root cause that nonetheless took developers a month to fix. Wu pointed out that a data race bug in a critical Web service that remains unresolved for weeks or months could be exploited by an attacker. As a solution, Wu proposed LOOM, a tool that can work around known race conditions in live programs before the developer releases a fix. LOOM accomplishes this by using execution filters, which are simple declarative statements that explicitly synchronize regions of application code.

LOOM combines static preparation with live updates. Specifically, a LOOM compiler plugin bundles the LOOM update engine into an application binary. A user wishing to install an execution filter on a live application invokes the LOOM controller. The LOOM controller is responsible for translating the filter's declarative statements into specific operations (e.g., mutual exclusion is translated into lock/unlock, ordering requirements are translated into semaphores, etc.). The controller passes the translated application code to the LOOM update engine responsible for patching the live application. To ensure that filters can be safely installed, Wu described an algorithm termed evacuation that uses static analysis to identify dangerous regions of code and ensures that (1) threads wishing to enter a dangerous region are paused at a safe location before the region, and (2) threads already executing in the dangerous region exit the region before the update is installed.

Wu described LOOM's evaluation on nine data races in six applications—in all cases, LOOM applied the live update in a timely manner and bypassed the race. On average, LOOM adds less than 5% performance overhead. Wu also described an alternate implementation using the Pin dynamic instrumentation tool, which introduced a 10x overhead. Wu demonstrated that LOOM is also scalable—MySQL running with 32 threads experiences only a 12% performance overhead.

Richard Draves (Microsoft Research) was curious if the evacuation algorithm makes the application susceptible to deadlock. Wu responded that while an application could deadlock, they did not come across any in their evaluation of LOOM. Further, LOOM implements a mechanism to uninstall the fix if a deadlock manifests. The next questioner asked if Wu could use information collected at runtime to fix the bug in the source code. Wu responded that users can examine the application source code and can also write execution filters targeted at specific file names/line numbers to fix the bug.

Effective Data-Race Detection for the Kernel

John Erickson, Madanlal Musuvathi, Sebastian Burckhardt, and Kirk Olynyk, Microsoft Research

John Erickson, a test lead on Microsoft's Windows 7 team, described the first data race in the Windows kernel he had discovered using DataCollider: a developer had assumed that certain bitwise operations were atomic without realizing that the operations were actually read/write byte instructions. A rare write-write interleaving would overwrite a callback flag resulting in a system hang. As this race was timing dependent, it was not caught prior to shipping Windows 7. This example illustrates that data races are hard to detect and debug.

The primary challenge in using existing happens-before and lockset algorithms to detect data races in the kernel is that both approaches need to be aware of all locking semantics. To address this challenge, Erickson proposed DataCollider, a data race detection tool for the kernel that is based on two insights: (1) instead of inferring a data race, cause a data race to actually manifest so there are no false positives; (2) the overhead of instrumenting all memory accesses can be significantly reduced by sampling a random subset of all memory accesses. DataCollider works as follows: a code or data breakpoint is set on a memory location being sampled. If a thread accesses the sampled memory location, it is blocked for a short time window. If a second thread tries to access the same memory location within that window without grabbing the appropriate lock, the breakpoint causes the second thread to block too. By pausing both threads at the instance the race occurs, DataCollider has access to both thread stacks, the full register and global state of which can be logged to report the race to the developer.

DataCollider allows users to control execution overhead by explicitly selecting the sampling rate—a sampling rate of 0% allows the code to execute at native speed. An evaluation of DataCollider on Windows 7 revealed 113 races on booting to the desktop. While the majority are benign, 25 are confirmed bugs in Windows 7 that are fixed or will be fixed soon.

George Candea (EPFL) wondered about the difficulty in categorizing races in real products as benign or malign. Erickson responded that much of the bucketization was manual and required source code analysis. Rik Farrow questioned whether breakpoints were truly randomly selected or whether they were applied at carefully selected locations. Erickson responded that all the memory accesses were enumerated and breakpoints were set on a random sampling without any knowledge of the actual memory accesses. The only tuning factor is that users can specify how many breakpoints they wish to execute per second. Michael Brodsky (MIT) asked how dependent DataCollider is on having debug symbols, since third-party driver manufacturers might not provide these with their binaries. Erickson responded that DataCollider requires debugging symbols, so breakpoints are applied to actual memory locations. Peter Druschel (MPI-SWS) wondered how the sampling time and detection rate were related to the total number of races. Erickson answered that while they do want to ensure that DataCollider is detecting a uniform sampling of races and not just the easiest 20, they cannot evaluate if this is the case, since no other data race detectors operate within the kernel.

Ad Hoc Synchronization Considered Harmful

Weiwei Xiong, University of Illinois at Urbana-Champaign; Soyeon Park, Jiaqi Zhang, and Yuanyuan Zhou, University of California, San Diego; Zhiqiang Ma, Intel

Concurrent programs increasingly rely on synchronization to guarantee correct execution. While many popular applications such as MySQL use standard synchronization libraries (e.g., pthreads), others such as LDAP rely on ad hoc synchronization, which is often harder to understand and debug.

The primary contribution of Xiong's work is quantitative evidence that ad hoc synchronization should be considered harmful. Xiong and his co-authors spent three months documenting every instance of ad hoc synchronization in 12 concurrent programs, including MySQL, Apache, and Mozilla. While all of them implemented some form of ad hoc synchronization, Xiong found that MySQL employed the most—83 instances. Next, Xiong analyzed bug patches uploaded to the Bugzilla database and discovered that almost half of all ad hoc synchronizations resulted in a concurrency bug. Unfortunately, existing static analysis tools that detect deadlocks, data races, and other concurrency bugs are rendered useless for multi-threaded programs that use ad hoc synchronization, as these tools do not understand the custom-locking semantics employed in ad hoc synchronization.

The second contribution of Xiong's work is SyncFinder, a tool that automatically detects and annotates ad hoc synchronization. The insight in SyncFinder is that every ad hoc synchronization executes in a loop with an exit condition that is controlled by a dependent variable. After applying several pruning algorithms, SyncFinder identifies shared dependent variables that control ad hoc sync loops and annotates all instances where the variables are read or written to. An evaluation of SyncFinder revealed that, on average, it detects 96% of all ad hoc synchronizations with a 6% false-positive rate.

The first questioner wondered how many bugs normal synchronization introduced in comparison to ad hoc synchronization. Xiong responded that the percentage of bugs is much higher in ad hoc synchronization. Bryan Ford (Yale) wondered what the current synchronization primitives lack that caused developers to turn to ad hoc synchronization. Xiong responded that, from an analysis of program comments, it appeared that developers just wanted a flexible and simple synchronization primitive. Ford wondered what such a synchronization primitive should offer, to which Xiong responded that such a primitive already existed: conditional waits. Dan Peek (Facebook) suspected that programmers implementing ad hoc synchronization wished to avoid a synchronous write on the system bus which hurts performance, and wondered whether such synchronization was always harmful. Xiong responded that while not always harmful, ad hoc synchronization might not scale with the underlying architecture, and also degenerates code maintainability. Dan Peek followed up by asking if lock-free data structures should also be considered harmful. Rather than commenting on lock-free data structures, Xiong offered that developers should use well-known existing synchronization primitives. Chris Hawblitzel (Microsoft Research) wondered whether developers declaring synchronization variables as volatile would help SyncFinder. Xiong responded that if C/C++ adopted volatile as a standard it would help SyncFinder. Josh Triplett (Portland State University) wondered how fast SyncFinder runs. Xiong responded that the largest codebase SyncFinder was executed on was MySQL, which has over 1 million lines of code and took about 2.5 hours. On the other hand, OpenLDAP, much smaller but containing a lot of ad hoc synchronization with significant data and control dependency, also took 2.5 hours.

Monster Poster Session

First set of posters summarized by John McCullough (jmccullo@cs.ucsd.edu)

VSSIM: Virtual SSD Simulator

Joohyun Kim, Haesung Kim, Seongjin Lee, and Youjip Won, Hanyang University, Seoul, Korea

VSSIM allows researchers to explore different configurations of SSD hardware. The simulator emulates both page and hybrid translation layers. The researchers have validated the expected behavior of higher block usage overhead of the hybrid translation layer as well as the effects of TRIM. Contact: james@ece.hanyang.ac.kr.

Energy Consumption Behavior of Modern SSD and Its Architectural Implication

Balgeun Yoo and Youjip Won, Hanyang University, Seoul, Korea

Yoo and Won evaluate the energy use of an X-25M SSD. Using writes of varying size, they find that energy use is proportional to the number of channels that are activated as well as the number of ways—chips within a single package—that are active. Changes in power draw are bounded between consumption for 8kB writes and 512kB writes. Contact: starthunter@ece.hanyang.ac.kr.

ABACUS: A Configurable Profiling Engine for Multicore Processors

Sergey Blagodurov, Eric Matthews, Sergey Zhuravlev, Lesley Shannon, and Alexandra Fedorova, Simon Fraser University

There are a number of hardware performance metrics that are useful for multicore scheduling that are not commonly available. ABACUS is a system that implements counters such as instruction mix and pipelines stalls with no processor time overhead on an FPGA-based processor core. Contact: sba70@cs.sfu.ca.

DoublePlay: Parallelizing Sequential Logging and Replay

Kaushik Veeraraghavan, Dongyoon Lee, Benjamin Wester, Peter M. Chen, Jason Flinn, and Satish Narayanasamy, University of Michigan

DoublePlay employs multiple executions to detect data races. The system uses a novel technique of dividing a parallel execution into epochs, recording the high-level thread execution schedule along with memory checkpoints. By re-executing the parallel schedule in a serial manner, the memory states can be compared to detect data races. Contact: kaushikv@umich.edu.

Using Program Analysis to Diagnose Configuration Problems

Ariel Rabkin and Randy Katz, University of California, Berkeley

Java has a wide variety of configuration parameters. Deciphering which parameter might be responsible for a stack trace can be challenging. This work uses static analysis to tie error points in Java to the relevant configuration parameters. Contact: asrabkin@eecs.berkeley.edu.

DCR: Replay Debugging for the Data Center

Gautam Altekar and Ion Stoica, University of California, Berkeley

Deterministic replay is very helpful in debugging distributed systems but can have high overhead. Most distributed system bugs arise from control plane errors. By logging only the control traffic, the total volume can be reduced by 99%. Thus, the control traffic can be used to replay the overall behavior and semantically equivalent data can be constructed using STP techniques. Contact: galtekar@cs.berkeley.edu.

Reconfigurable Virtual Platform for Real Time Kernel

Dilip K. Prasad, Nanyang Technological University, Singapore; Krishna Prasath, Coventry University, UK

Evaluating real-time applications across multiple platforms can be very challenging. Prasad and Prasath have created a platform that is easily reconfigurable for different operating systems and that cleanly integrates into an IDE. Contact: dilipprasad@pmail.ntu.edu.sg.

ErdOS: An Energy-Aware Social Operating System for Mobile Handsets

Narseo Vallina-Rodriguez and Jon Crowcroft, University of Cambridge

ErdOS leverages social interactions to improve energy use in mobile devices. For instance, if a power-hungry application is redundant to a localized area, a single device can take the measurement and gossip the results to those allowed by social graph access controls. Contact: nv240@cam.ac.uk; Web: http://www.cl.cam.ac.uk/~nv240/erdos.html.

Leviathan: Taming the #ifdef Beast in Linux et al.

Wanja Hofer, Christoph Elsner, Frank Blendinger, Wolfgang Schröder-Preikschat, and Daniel Lohmann, Friedrich-Alexander University Erlangen-Nuremberg

Understanding ifdef-laden code can be very challenging in many situations. The authors have created a pre-processing FUSE plug-in that allows developers to operate on the pre-processed code. Edits to either the view or the code are automatically transferred back to the appropriate part of the original. Contact: hofer@cs.fau.de.

Joan: Shepherd Application Privacy with Virtualized Special Purpose Memory

Mingyuan Xia, Miao Yu, Zhengwei Qi, and Haibing Guan, Shanghai Jiao Tong University

Multiple network applications executing on the same machine are not entirely safe from one another. The authors explore a technique of per-application hypervisor protected memory that can be selectively shared in either modifiable or read-only form.

Configuration Bugs in Linux: The 10000 Feature Challenge

Reinhard Tartler, Julio Sincero, Wolfgang Schrîder-Preikschat, and Daniel Lohmann, Friedrich-Alexander University Erlangen-Nuremberg

The configuration specified in the Linux config file may not be correctly embodied by the relevant ifdefs in the code. Misspellings and contradictions from the combination of ifdef conjunctions and feature dependencies can lead to dead or misbehaving code. The authors have used code analysis to identify many points of contradictory and dead code, culminating in 123 patches with 64 acknowledged by kernel authors. Contact: tartler@informatik.uni-erlangen.de.

Backup Metadata as Data: DPC-tolerance to Commodity File System

Young Jin Yu, Dong In Shin, Hyeong Seog Kim, Hyeonsang Eom, and Heon Young Yeom, Seoul National University

File system metadata is critical to the integrity of the actual data stored on the file system, but it is ignored by typical backup techniques. The authors extract the filesystem-level pointers for use in a filesystem recovery mechanism that is faster than traditional file system scans.

Second set of posters summarized by Robert Soule (soule@cs.nyu.edu)

Using Mobile Phones to Set Up Secure VNC Sessions on Kiosks

Wooram Park, Sejin Park, Baegjae Sung, and Chanik Park, Pohang University of Science and Technology

Many people use publicly available computers called kiosks to access remote desktops. In order to prevent the leakage of private data, this work proposes using a mobile phone to establish a VNC session with the remote desktop. The kiosk attests its identity to the remote desktop, using keys issued by a remote key server, which are stored on the mobile phone.

Aggressive VM Consolidation with Post-Copy-based Live Migration

Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi, National Institute of Advanced Industrial Science and Technology

There are two techniques for migrating live virtual machines, pre-copy and post-copy. This work implements the post-copy live virtual machines migration of KVM, which performs significantly faster than previous pre-copy implementations.

Mnemosyne: Lightweight Persistent Memory

Haris Volos and Michael Swift, University of Wisconsin—Madison; Andres Jaan Tack, Skype Limited

Storage class memory provides low-latency, persistent storage. This work seeks to provide programmers with direct access to storage class memory. Mnemosyne provides two abstractions: one for allocating memory, and a second, called durable memory transactions, for atomically modifying persistently stored data structures.

Beacon: Guiding Data Placement with Application Knowledge in Multi-Tiered Enterprise Storage System

Hyojun Kim, Georgia Institute of Technology; Sangeetha Seshadri, IBM Almaden Research Center; Yi Xue, IBM Toronto; Lawrence Chiu, IBM Almaden Research Center; Umakishore Ramachandran, Georgia Institute of Technology

Enterprise storage systems contain both cheap but slow HDD and fast but expensive SSD. This work tackles the problem of deciding what data is stored in which storage system. Current approaches use a reactive approach, which places data after monitoring usage over time. In contrast, this work modifies applications to provide hints about their spatial (what files), temporal (duration), and priority requirements, allowing for predictive placement of data.

Guest Transparent Dynamic Memory Balancing in Virtual Machines

Changwoo Min, Inhyuk Kim, Taehyoung Kim, and Young Ik Eom, Sungkyunkwan University

Multiple guest operating systems on a single host machine compete for memory. This work seeks to solve those conflicts by estimating the memory requirements of each guest operating system and pre-allocating that memory.

Parallel Operating System Services in fos

David Wentzlaff, Charles Gruenwald III, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Harshad Kasture, Lamia Youseff, Jason Miller, and Anant Agarwal, Massachusetts Institute of Technology

fos is an operating system for multicore systems that treats operating system services like distributed Internet servers. Each service is implemented as a set of spatially separated server processes called a fleet. The servers' processes in a fleet collaborate to provide both high-level services and as low-level services such as page allocation, scheduling, and memory management.

S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems

Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

S2E provides an automated path explorer and modular path analyzers and is used for various tasks, including performance profiling, reverse engineering software, and bug finding in both kernel-mode and user-mode binaries. S2E scales to larger real systems, such as a full Windows stack, better than prior work.

Consistent and Durable Data Structures for Non-Volatile Byte-Addressable Memory

Shivaram Venkataraman, University of Illinois; Niraj Tolia, HP Labs; Roy H. Campbell, University of Illinois

Hardware manufacturers have developed new nonvolatile memory devices. This work explores what are the best data structures for storing data on these new devices. Rather than using the traditional write-ahead logging approach, this work keeps multiple copies of the data structures (for example, a B-tree) and atomically moves between subsequent versions.

Tracking and Exploiting Renewable Energy in Grid-Tied Datacenters

Nan Deng and Christopher Stewart, Ohio State University

Datacenters use a device called a grid-tie to combine energy from renewable sources and energy supplied by a grid. When placing these grid ties in the datacenter, engineers must make a choice between placing a small number of high-capacity grid-ties (which themselves consume energy), or a large number of low-capacity grid-ties. This work seeks to find the optimal placement strategy through simulation.

Making Tracks with Malware: Control Flow Visualization and Analysis

Jennifer Baldwin, University of Victoria

This work presents a new tool for visualizing the control flow of assembly programs. The tool provides better visualizations than existing systems, and demonstrates the clarity of its presentation by showing the control flow of a malware application.

dBug: Systematic Evaluation of Distributed Systems

Randy Bryan, Garth Gibson, and Jiri Simsa, Carnegie Mellon University

dBug is a tool for finding bugs in distributed systems. It introduces a thin interposition layer between the application and the operating system, which hijacks system calls for synchronization and shared memory. An arbiter scheduler then systematically explores different execution paths to discover bugs.

Dynamic Forwarding Table Management for High-speed GPU-based Software Routers

Joongi Kim, Keon Jang, Sangjin Han, KyoungSoo Park, and Sue Moon, KAIST

In prior work, the author implemented a software router in a GPU. After using the system in practice, it became clear that there was a need for dynamic updates to the forwarding tables. This work explores how to support these dynamic table updates in the presence of bursty network traffic.

Preventing Memory-Allocation Failures Through Anticipatory Memory Allocation

Swaminathan Sundararaman, Yupu Zhang, Sriram Subramanian, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Recovery code for memory allocation failures in file systems is buggy. This work seeks to avoid executing that recovery code at all. A utility performs static analysis on the file system code to determine how much memory the software will request. The system then pre-allocates the memory and proxies all subsequent requests for memory, servicing the requests from the pre-allocated pool.

Third set of posters summarized by Michael Roitzsch (mroi@os.inf.tu-dresden.de)

Coerced Cache Eviction: Dealing with Misbehaving Disks through Discreet-Mode Journalling

Abhishek Rajimwale, Vijay Chidambaram, Deepak Ramamurthi, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

What if the controller of a magnetic hard drive or an SSD does not fully honor cache flush requests for its internal cache? If you run a workload that tries to enforce consistency requirements by explicitly flushing the cache, a system crash may lead to unexpected inconsistencies. The poster authors suggest forcefully evicting everything from the cache by sending a cache flush workload to the disk. They address the problem of balancing the resulting overhead with the probability of achieving a full flush.

Testing Device Drivers without Hardware

Matthew J. Renzelmann, Asim Kadav, and Michael M. Swift, University of Wisconsin—Madison

The authors want to simplify driver maintenance for kernel developers without access to the device in question. This situation occurs when making changes to kernel interfaces that lead to subsequent changes in other drivers. The proposed solution consists of a symbolic execution engine for the driver code, combined with a symbolic hardware representation. This symbolic hardware is generic per device-class. KLEE is used for the symbolic execution. To avoid symbolic execution of startup code like kernel boot and device initialization, the system uses device traces to fast-forward to the interesting parts of the execution.

Spark: Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury, Justin Ma, Michael J. Franklin, Scott Shenker, and Ion Stoica, University of California, Berkeley

A new programming model for applications processing large amounts of data in a distributed environment was presented. Whereas MapReduce and Dryad are well fitted for acyclic data-flow algorithms, this model focuses on algorithms that reuse a working set across operations. One target application is interactive data mining. The programming paradigm is constructed around the concept of a Resilient Distributed Dataset (RDD). An RDD provides an abstraction for objects that handles distribution and persistence. To achieve the fault-tolerance of MapReduce, an RDD can be rebuilt if it is lost due to failure.

The CONSCIOUS Virtual Machine Model: Transparently Exploiting Probability Processors

Jonathan Appavoo, Dan Schatzberg, and Mark Reynolds, Boston University; Amos Waterland, Harvard University

When collecting instruction pointer traces of a virtual machine, the authors noticed recurring patterns in the trace. They proposed the idea of collecting a repository of those patterns and the results of the operations they represent. If such a known pattern is recognized in the trace of a running VM, the execution can fast-forward to the end of the pattern and apply the canned result. This can save time and energy by not repeating recurring operations. As a side effect, the VM traces can be converted to audio files so you can actually hear the patterns!

Tracking and Exploiting Renewable Energy in Grid-Tied Datacenters

Nan Deng and Christopher Stewart, Ohio State University

Datacenters can be equipped with on-site renewable energy generators such as wind turbines or solar collectors. However, the energy provided by those does not suffice to run the datacenter, so energy from the traditional power grid needs to be mixed in. This task is performed by a grid-tie component which itself consumes energy and can fail, so the number and placement of these components poses an interesting research challenge.

Dynamic Runtime Optimizations inside BT-based VMMs

Mehul Chadha and Sorav Bansal, Indian Institute of Technology Delhi

Traditional runtime optimization of an application is done by tracing and just-in-time compilation of the application. The presenter, Sorav Bansal (sbansal@cse.iitd.ernet.in), proposes extending this idea to the entire system. Combining a virtual machine monitor with binary translation and a just-in-time compiler, runtime optimizations can be performed throughout the system, even crossing protection boundaries. The system can be enhanced with peephole and trace-based optimizations in future work.

Can You Keep a Secret?

David Cock, NICTA and University of New South Wales

Covert timing channels are an increasingly relevant problem for crypto systems, as the recent padding oracle attack on ASP.NET-based Web sites has shown. The poster presenter,

David Cock (david.cock@nicta.com.au) proposed using a real-time scheduler on the seL4 microkernel to control component response times. The cryptographic code of an application, like the OpenSSL library, would be separated into an isolated component with individual scheduling. Hooking into seL4's communication endpoints, the scheduler can delay responses of a component and thus decouple the component's actual response time and the behavior observable from outside. Reducing the variation on observed response time, the scheduler can reduce the bandwidth of the timing channel.

The Private Peer Sampling Service: The Ground for Your Secret Society

Valerio Schiavoni, Etienne Rivière, and Pascal Felber, University of Neuchâtel, Switzerland

The subject of this poster is group membership within a larger peer-to-peer overlay network. The network is cryptographically protected by the exchange of public keys. If you want to form groups within this network, you want to keep the group membership information private. The work presented on the poster employs onion routing and attacks the problems of firewalls, NATs, and the resulting challenges regarding the visibility of nodes.

Gang scheduling isn't worth it . . . yet.

Simon Peter, Andrew Baumann, and Timothy Roscoe, ETH Zurich

Andrew Baumann from the Barrelfish group (http://www.barrelfish.org/) presented this poster. Looking at opportunities for gang scheduling in the Barrelfish multikernel, the team studied the merits of gang scheduling for different workloads. Today's commodity operating systems typically do not employ gang scheduling. But this usually does not hurt, because workloads on these systems are dynamic, bursty, and interactive. Multiple parallel applications with fine-grained synchronization would be needed for the lack of gang scheduling to become a problem. One situation where it would help are stop-the-world garbage collectors.

Multi-pipes

Eric Van Hensbergen, IBM Research; Noah Evans, Alcatel Lucent Bell Labs; Pravin Shinde, ETH Zurich

This poster presented one aspect of a larger project on operating systems and services for high performance computing that scale to millions of cores. Specifically, it presented a data flow abstraction developed for BlueGene supercomputers. Multi-pipes are a generalization and extension of traditional UNIX pipes. Multiple readers and writers are handled deterministically, including fan-in and fan-out scenarios. Operations like broadcasts or reductions on the data traveling through the multi-pipe are supported. As a bonus, shell integration for multi-pipes is available.

Scaling Middleboxes through Network-Wide Flow Partitioning

Norbert Egi, Lancaster University; Lucian Popa, University of California, Berkeley; Laurent Mathy, Lancaster University; Sylvia Ratnasamy, Intel Labs, Berkeley

Large network installations contain many routers and each one contains a general-purpose processor. Turning those routers into middleboxes lets them perform duties like policing traffic, caching, or traffic encryption. However, this functionality is not now in one central location like a traditional firewall, but distributed throughout the network. Traffic must be suitably partitioned to scale this distributed middlebox system.

STEAMEngine: Driving the Provisioning of MapReduce in the Cloud

Michael Cardosa, Piyush Narang, and Abhishek Chandra, University of Minnesota; Himabindu Pucha and Aameek Singh, IBM Research—Almaden

STEAMEngine is a runtime for assigning VMs to execute MapReduce workloads. Given a set of VMs distributed over multiple machines, pending MapReduce tasks are mapped to a subset of those VMs. Individual jobs are profiled at runtime to model their resource needs. The spatio-temporal assignment of MapReduce jobs to VMs can then be tuned to meet timeliness or energy goals.

KÁRMA: A Distributed Operating System for MicroUAV Swarms

Peter Bailis, Karthik Dantu, Bryan Kate, Jason Waterman, and Matt Welsh, Harvard University

Studies show the US bee population decreased by 30%. But agriculture needs bees to pollinate the crops. The resulting gap is closed with the Robobees: minimal robotic bees that include sensors and compute resources. The particular aspect presented in this poster is the development of a distributed system that exhibits swarm intelligence. Combining a suitable swarm programming model, the necessary hardware and artificial intelligence, Robobees can one day actually fly and perform the pollination tasks of a real bee.

Fourth set of posters summarized by Justin Seyster (jseyster@cs.stonybrook.edu)

Diagnosing Performance Changes by Comparing System Behaviours

Raja R. Sambasivan, Carnegie Mellon University; Alice X. Zheng, Microsoft Research; Elie Krevat, Michael Stroucken, William Wang, Lianghong Xu, and Gregory R. Ganger, Carnegie Mellon University

Sambasivan presented Spectroscope, a project that uses end-to-end traces to diagnose distributed system performance changes. Each trace represents the entire path of a request through the system. Comparing traces from before and after a performance change can help find the cause of the change. A comparison can find structural changes, such as a request being routed to a remote datacenter, and response-time changes in individual components. Both kinds of change are ranked by their contribution to the overall performance change so that the developer can localize the source of the problem. The authors used Spectroscope to diagnose several real-world performance bottlenecks in Ursa Minor, a prototype distributed storage system.

The Anzere Personal Storage System

Oriana Riva, Qin Yin, Dejan Juric, Ercan Ucan, Robert Grandl, and Timothy Roscoe, ETH Zurich

Oriana Riva described Anzere, which manages synchronization of media, contacts, and other personal data across multiple personal devices and virtual cloud resources. Anzere emphasizes the expressiveness of replication policies, which are specified with a logic constraint language. For example, the user can require that one-day-old music is accessible to a cell phone with no more than 100ms delay. As another example, the user can also set a limit on cloud storage usage in terms of a maximum monthly fee. Although potentially complex, these policies remain tractable.

Dynamic Resource Allocation in Computing Clouds using Distributed Multiple Criteria Decision Analysis

Yagz Onat Yazr and Chris Matthews, University of Victoria; Roozbeh Farahbod, Defense R&D Canada Valcartier; Stephen Neville, University of Victoria; Adel Guitouni, Defense R&D Canada Valcartier; Sudhakar Ganti, Yvonne Coady, and Burak Martonalt, University of Victoria

Yagiz Onat Yazir presented this poster about moving to a model of provisioning virtual machines with loosely defined resources instead of asking customers to specify a priori CPU, memory, and bandwidth requirements. This would make cloud computing services more like the public utilities that we are used to. In the case of the cloud, loosely defined resource requirements map to "enough" CPU, memory, and bandwidth to achieve a set response time for requests under the current load.

In this kind of model, virtual machine resource limits vary dynamically, which means it is neither efficient nor realistic to statically allocate virtual machines to physical nodes. VMs must migrate between nodes when a node becomes over- or underutilized because VM resource constraints changed.

Centralized VM allocation algorithms can produce a near-optimal configuration, but they require far too many migrations to be practical. The Multiple Criteria Decision Analysis technique (or PROMETHEE method) presented in this poster deals individually with problem nodes in order to limit migrations.

The allocater migrates VMs whenever an anomaly—an over- or underutilized node—becomes apparent. To deal with overutilization, it migrates VMs away from the node until it is correctly provisioned, and to deal with underutilization, it evacuates the node so it can shut down. The resulting VM allocations do not distribute resources as effectively as centralized provisioning, but they allow resource constraints to vary without an impractical amount of VM migrations.

Remote Desktops Using Streaming VM Record/Replay

Nitesh Mor, Shikhar Agarwal, Sorav Bansal, and Huzur Saran, Indian Institute of Technology Delhi

In traditional remote desktop systems, a lot of bandwidth is dedicated to sending images of the remote system's desktop back to the client.

The system presented in this poster instead transfers virtual machine replay information, which often requires substantially less bandwidth than desktop graphics. The remote machine runs on virtual machine software that supports virtual record and replay. All the recorded replay information (such as interrupt timings) is sent over the network to a virtual machine running on the client. The client virtual machine can replay the remote machine's execution in real time, as it receives the replay data.

The most dramatic speed improvement comes from the bandwidth reduction when playing a DVD, which over a traditional remote desktop connection requires streaming the DVD video itself. Streaming video is not necessary with VM record/replay, but initial state of both VMs must be

synchronized, meaning that the DVD's contents are already available at the client side. Because of the synchronization requirement, which involves an initial disk transfer of the VM, the technique is most suitable for a client making frequent remote desktop sessions to the same remote VM, or to multiple VMs which share similar disk images.

Porting File System Structures to Nameless Writes

Yiying Zhang, Leo Prasath Arulraj, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Commercial flash drives keep a mapping table from logical block addresses to physical block addresses, which the drive can use to dynamically remap blocks for the sake of wear leveling. Yiying Zhang explained that these tables become more wasteful as drives get bigger: a 2TB SSD would need 30GB of mapping tables for a page-level mapping using a standard flash page size. Although mapping strategies exist that can keep smaller mapping tables, they have a higher I/O cost.

Zhang and co-authors show how device support for "nameless writes" makes it possible to modify file systems to operate efficiently without device remapping and without modifying on-disk structures. A nameless write gives the disk the task of choosing a free block to write to. Because the disk chooses which blocks to write, it no longer needs to remap blocks to effectively spread out writes. The nameless write operation returns the physical block index to the file system.

To handle blocks that must be relocated after wearing out, the disk will also need to support a device callback, notifying the file system that it has to update pointers to the relocated block.

DiCE: Predicting Faults in Heterogeneous, Federated Distributed Systems

Vojin Jovanović, Marco Canini, Gautam Kumar, Boris Spasojević, Olivier Crameri and Dejan Kostić, EPFL, Switzerland

Marco Canini and Dejan Kostić presented DiCE, designed to analyze heterogeneous, federated distributed systems such as BGP, the Internet's routing protocol. DiCE can take a complete snapshot of all the nodes in the analyzed system and then explore possible system behaviors from that snapshot using "concolic" execution, a hybrid of symbolic and concrete execution. In BGP, this approach can find configuration errors that might allow prefix hijacking or policy conflicts. Fuzz testing makes it practical to search for valid messages that have the potential to harm the system.

Fifth set of posters summarized by Edmund L. Wong (elwong@cs.utexas.edu)

A Replay-based Approach to Performance Analysis

Anton Burtsev, Eric Eide, and John Regehr, University of Utah

Anton Burtsev presented a novel way of approaching performance analysis of complex software systems with a full-system deterministic replay. Burtsev leveraged Xen VMM to record the entire execution history of a system with a constant overhead of several percent. An analysis algorithm is invoked on a copy of the original execution, which is recreated by means of replay mechanisms offline. Burtsev argued that such an approach turns performance, a dynamic property of a particular execution, into a static property that can be analyzed separately from an actual instance of a running system. To provide general support for replay-based performance analyses, Burtsev suggested a general analysis framework which combines traditional replay mechanisms with a realistic performance model, and a semantic interface to the behavior and the runtime state of the system.

Mitigating Risk in Green Cloud Computing

Sakshi Porwal, Muki Haklay, John Mitchell, Venus Shum, and Kyle Jamieson, University College London

Sakshi Porwal and co-authors studied how power consumption, an important consideration for cloud computing providers, can be reduced. Porwal showed two models of computing: the waterfall model, in which tasks are assigned to an idle machine only after all other non-idle machines were fully utilized, and the distributed model, in which tasks are load-balanced across multiple machines. She showed that the waterfall model reduces power consumption and produces less CO2 as a result. Porwal also explored distributing tasks to sites which are located in geographically colder areas and thus require less power for cooling mechanisms, which represent approximately a third of power consumption at datacenters.

It Wasn't My Fault: Understanding OS Fault Propagation Via Delta Execution

Cristiano Giuffrida, Lorenzo Cavallaro, and Andrew S. Tanenbaum, Vrije Universiteit, Amsterdam

Cristiano Giuffrida proposed characterizing faults in OSes by introducing faults into the system, isolating the faulty execution, and comparing it to a fault-free execution to see how the execution, state, and environment differed. By creating a catalog of faults and their effects, Giuffrida envisioned that an OS can be developed that can recover from failures and continue execution. This catalog could be used to remove

traces of faulty behavior without affecting correct parts of the execution.

Fine-grained OS Behavior Characterization

Lorenzo Cavallaro, Cristiano Giuffrida, and Andrew S. Tanenbaum, Vrije Universiteit, Amsterdam

Lorenzo Cavallaro proposed building a microkernel-based OS with extremely focused and small components. The behavior of these components could be automatically profiled and characterized by using an IPC-based monitoring scheme. These profiles could then be later used to detect anomalous behavior and bugs in OS components. Cavallaro argued that the overhead of such a system could be ameliorated by giving users the option to adjust the granularity and accuracy of the monitoring infrastructure.

Resurrecting Static Analysis: Exploiting Virtualization to Address Packing and Encrypted Malware

Christopher Benninger, Niko Rebenich, and Stephen W. Neville, University of Victoria; Rick McGeer, HP Labs; Yvonne Coady, University of Victoria

Christopher Benninger proposed a new technique for detecting malware that may be "packed" (encrypted or compressed). Instead of relying on static techniques alone, which are difficult and often fail when malware is packed, or dynamic techniques, which are not as successful as static techniques before the advent of packing, Benninger argued that malware can be more accurately detected by exposing malware to static analysis after it has unpacked itself and right before it attempts to execute. Toward this goal, he developed an event-driven platform for identifying packed malware running on a VM, an introspection tool for accessing an offending process's memory space from a separate VM, and an analysis tool for identifying and flagging malware variants.

Block-IO Scheduling for Guaranteed Scalable Storage Performance

Bader Al Ahmad, Sriram Murali, and Sathish Gopalakrishnan, University of British Columbia

Sriram Murali observed that many applications require real-time interactivity, yet many of them run on cloud environments consisting of virtual machines handling multiple clients. Murali argued that disk I/O is the major barrier to achieving guaranteed QoS for virtual machines; thus, efficient disk utilization is critical to providing real-time guarantees. Towards this goal, Murali implemented a scheduler based on weighted fair-queuing in the block device driver of the Xen hypervisor which offers hard and soft real-time guarantees to the different virtual machines accessing the disk. This scheduler is scalable from commodity servers to enterprise cloud-based solutions.

SlapChop: Automatic Microkernels

Sara Dadizadeh, Jean-Sébastien Légaré, and Andrew Warfield, University of British Columbia

Modern software is complex and monolithic. Sara Dadizadeh presented SlapChop, a system for decomposing a large system into small, single-purpose source code libraries that perform specialized tasks. SlapChop dynamically analyzes a running system, collects traces, finds the parts that are relevant to the task that is to be isolated, maps those instructions back to the original source, and generates specialized libraries consisting of this source code. She showed an example of SlapChop being performed on an OS kernel.

Benchmarking Online Storage Providers—Does Bandwidth Matter?

Andi Bergen, University of Victoria; Rick McGeer, HP Labs; Justin Cappos, University of Washington; Yvonne Coady, University of Victoria

Andi Bergen argued for the importance of benchmarking online storage providers, which are becoming increasingly popular. Because users are often bandwidth-limited in their own connections, Bergen argued that such measurement should be done in a distributed fashion, by having multiple sites perform many operations on the storage providers. His hope is to develop a tool that allows for customizable benchmarking depending on what metrics users are interested in.

Measurements of Personally Identifiable Information Exposure on the Web

Xiao Sophia Wang, University of Washington; Sam Burnett, Georgia Tech; Ben Greenstein, Intel Labs Seattle; David Wetherall, University of Washington

Personally identifiable information (PII), such as names, addresses, and credit card numbers, are being used online as a part of the Web, yet very little is known about how prevalent PII exposures are in practice. Xiao (Sophia) Wang showed that, although users would expect that PII is only sent to the Web sites they are visiting, this information is often sent to third-parties or easily gleaned by eavesdropping or through cookies. Studying the top 100 Web sites in the US, Wang, surprisingly, found that 35% send passwords in the clear, 26% send some form of PII to third parties, and 54% store some form of PII in cookies.

Depot: Cloud Storage with Minimal Trust

Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish, University of Texas at Austin

Sangmin Lee and Srinath Setty presented Depot, a cloud storage system that minimizes trust for safety and for liveness. Depot achieves this through a new consistency model, fork-join-causal consistency, which guarantees causal consistency in the absence of faults and handles forks by reducing equivocation to concurrency. Despite its minimal reliance on trust, Depot still provides useful properties: safety properties include eventual consistency, fork-join-causal consistency, bounded staleness, integrity, recoverability, and eviction of faulty nodes. Liveness properties include allowing nodes to always write, always exchange updates, and read when correct nodes have the required objects. Lee and Setty argued that Depot's cost is low in terms of latency, and its weighted dollar cost is modest.

Iodine: Interactive Program Partitioning

Nathan Taylor and Andrew Warfield, University of British Columbia

Clearly identifying important parts of modern software is complicated but imperative for understanding whether exploits or bugs may exist in the code. Nathan Taylor described his system, Iodine, as a way to discover these components through techniques inspired by MRI. Iodine outputs a control-flow graph with edges representing control-flow connections and colors representing how frequently a particular component is visited. These colors eventually fade if a component is no longer visited, giving the user an interactive idea regarding what code is being executed over time. The system provides users with an interactive interface, allowing users to poke into the code and explore the effects that unvisited branches have on the graph.

Masking Network Heterogeneity with Shims

Danny Yuxing Huang, Williams College; Eric Kimbrel and Justin Cappos, University of Washington; Jeannie Albrecht, Williams College

Danny Yuxing Huang tackled the problem of dealing with heterogeneous network environments—those which contain NATs, VPNs, firewalls, mobile devices, etc.—by developing a new technique for building applications. A common solution involves using libraries to virtualize network heterogeneity. Huang argued that porting programs with these libraries to deal with specific types of network heterogeneity is tedious and error-prone; due to semantic differences, the programs and/or the libraries must be modified to work together. Instead, Huang proposed the use of formally verified wrappers, or shims, that abstract away the details of providing functionality such as NAT traversal or mobility support from applications. Unlike the above-mentioned libraries, shims use an interface that is semantically identical to the underlying network interface. Thus, shims are completely transparent to the application, require no modification of the original code, and may be composed together dynamically. Huang demonstrated his technique by constructing a NAT traversal shim that enables an unmodified server application to accept incoming connections from behind a NAT device.

Reducing System Call Latency via Dedicated User and Kernel CPUs

Josh Triplett, Philip W. Howard, Eric Wheeler, and Jonathan Walpole, Portland State University

Systems have an increasing number of cores, allowing tasks to often have their own CPUs. Josh Triplett observed, however, that tasks still time-slice between user and kernel modes. Triplett proposed leveraging the increasing number of cores by providing each application with a dedicated syscall thread, which stays in-kernel. This thread has its own CPU and performs the actual system calls, in order, on behalf of application threads. Triplett argued that his approach does not require modification of user code or existing kernel syscalls and does not require a new type of thread or scheduler, while improving the performance of common syscalls in Linux.

Sixth set of posters summarized by William Enck (enck@cse.psu.edu)

Deterministic Concurrency within the OpenMP Framework

Amittai Aviram and Bryan Ford, Yale University

Available techniques to prevent shared-memory concurrency bugs are insufficient; developers infrequently adopt new languages, and deterministic thread schedulers make race conditions repeatable without eliminating them. Amittai Aviram (amittai.aviram@yale.edu) presented Deterministic OpenMP (DOMP), which simplifies the prevention of race conditions by following a programming model resembling document handling in version control systems. When concurrent threads start executing at a fork, DOMP makes a copy of the current shared program state for each thread. Once the threads finish and rejoin their parent, DOMP merges their writes into the parent's state; if two threads have written to the same memory location, DOMP signals the race condition as an error. DOMP will implement most of the core features of standard OpenMP, including the *parallel, for,* and *sections* directives, as well the *reduction* clause, thus making it easy to parallelize existing sequential code in conventional languages deterministically by adding just a few annotations.

Diagnosing Intermittent Faults Using Software Techniques

Layali Rashid, Karthik Pattabiraman, and Sathish Gopalakrishnan, University of British Columbia

Layali Rashid (lylrashid@gmail.com) explained that today's complex hardware chips are prone to intermittent errors. When errors are detected, faulty components should be disabled. Given the existence of an error, this work tracks program dependencies to the instructions where the error originated. Using the properties of the offending instruction, the faulty logic unit is diagnosed and disabled to ensure that future intermittent errors cannot affect program correctness.

NanoXen: Better Systems Through Rigorous Containment and Active Modeling

Chris Matthews, University of Victoria; Justin Cappos, University of Washington; Yvonne Coady, University of Victoria; John H. Hartman, University of Arizona; Jonathan P Jacky, University of Washington; Rick McGeer, HP Labs

Current systems lack robustness and security. This work, presented by Chris Mathews (cmatthew@cs.uvic.ca), proposes a new computational model based on "virtual components." In this model, applications execute as sets of components. Components are primitive computational units with well-defined semantics. The components execute inside of either "native client" (NaCl) containers or virtual machine (VM) containers, depending on isolation requirements. While this work is at a very preliminary state, Matthews and his co-authors aim to use these primitives to design a more robust and secure operating system environment.

Namespace Composition for Virtual Desktops

Dutch Meyer, Mohammad Shamma, Jake Wires, Maria Ivanova, Norman C. Hutchinson, and Andrew Warfield, University of British Columbia

Dutch Meyer (dmeyer@cs.ubc.ca) explained that many enterprises are beginning to deploy PCs as thin clients running virtual machines served from back-end servers. These systems generally operate at the block level, which is semantically poor and difficult to administer. Meyer and his co-authors proposed a model that serves file namespaces instead of raw block devices. The proposed system can create namespaces for new VMs on the fly and can merge namespaces from multiple clients to create new views of the system helpful for collaboration and administration. By focusing on file semantics, redundant storage and scanning can be eliminated, and overall management complexity can be reduced.

Compression of High Resolution Climate Data

Jian Yin and Karen Schuchardt, Pacific Northwest National Lab

Climate simulations consume and produce petabytes of data. Traditional compression algorithms perform poorly on this data and frequently require the entire archive to be decompressed before being used by simulations and analysis tools. Jian Yin (jian.yin@pnl.gov) and his co-authors developed a compression algorithm based on properties of the climate data that allows block-based decompression for use of subsets of the overall dataset. They use heuristics to determine where to break the compression blocks such that the decompression and analysis can be pipelined. Additionally, the decompressed blocks are cached to avoid redundant decompression and speed analysis. Finally, they include prediction algorithms to decompress data blocks so that new blocks are immediately available.

Informed System Design through Exact Measurement

Daniel A. Freedman, Tudor Marian, Jennifer Lee, Ken Birman, Hakim Weatherspoon, and Chris Xu, Cornell University

Daniel A. Freedman (dfreedman@cs.cornell.edu) considered the application of a class of exact network measurements to help inform system design, particularly for architectures that involve the intersection of endpoint systems and network links. He discussed the design of network instrumentation—using physics test equipment, such as oscilloscopes, pattern generators, lasers, etc.—for the exact capture of packets in flight, and they demonstrate its application for a particular deployed 10 Gigabit Ethernet wide-area network (WAN). In fact, on such a WAN, they observe anomalous behavior that contests several common assumptions about the relationship between input and output traffic flows. Finally, the authors connect their observations of emergent packet chains in the network traffic with higher-level system effects—namely, to explain previously observed anomalous packet loss on receiver endpoints of such networks.

Accele Scheduler: Energy Efficient Virtual CPU Scheduling for Modern Multicore CPUs

Tetsuya Yoshida and Hiroshi Yamada, Keio University; Hiroshi Sasaki, University of Tokyo; Kenji Kono, Keio University; Hiroshi Nakamura, University of Tokyo

CPU chips now frequently include multiple processing cores, but not all of the cores always need to run at their highest frequency. Dynamic voltage and frequency scaling (DVFS) improves energy efficiency by scaling down voltage and frequency when possible. On modern multicore chips, all cores must use the same voltage, due to architectural limitations, even if their frequencies are individually scaled. Tetsuya Yoshida (tetsuyay@sslab.ics.keio.ac.jp) and his co-authors

have developed a scheduling algorithm for virtual CPUs that distributes the processing load to optimize energy consumption in such an environment. They have already achieved an energy delay product (EDP) of 23.6%, which is better than Xen's existing credit scheduler.

Redflag: Detailed Runtime Analysis of Kernel-Level Concurrency

Justin Seyster, Abhinav Duggal, Prabakar Radhakrishnan, Scott D. Stoller, and Erez Zadok, Stony Brook University

Justin Seyster (jseyster@cs.stonybrook.edu) explained that large code bases, such as the Linux kernel, are difficult to analyze for concurrency bugs. Seyster and his co-authors have developed a runtime analysis system to focus on the locking of any specific data structure. The tool provides lockset and block-based analysis. When working with code as large and complex as the Linux kernel, subtle complications result. For example, occasionally variables contain bit-fields to store system state. Therefore the tool must treat individual bits in such variables as variables themselves. The tool is also sensitive to the order in which locked blocks execute. Using their tool, Redflag, they have identified one race condition in their own file system code and independently discovered two additional known-concurrency issues in Linux's file system code. The authors hope to apply the analysis to other data structures within the Linux kernel.

Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns

Etienne Le Sueur and Gernot Heiser, University of New South Wales

Etienne Le Sueur (etienne.lesueur@nicta.com.au) and Gernot Heiser observed the power consumption of three multicore AMD Opteron systems over a seven-year period to study the effectiveness of dynamic voltage and frequency scaling (DVFS). They found four factors that cause DVFS to become significantly less effective: (1) scaling in semiconductor technology, (2) increased memory performance, (3) improved sleep/idle modes, and (4) multicore processors. They believe that in the future, cache management techniques such as turning of parts of the L3 cache, which is as large as 8MB, will be much more effective than DVFS at conserving energy.

Research Vision Session

Summarized by Shivaram Venkataraman (venkata4@illinois.edu)

The research vision session held at OSDI '10 was similar in spirit to the Wild and Crazy Ideas sessions at ASPLOS. It consisted of four presentations on systems research ideas for the future. Ice cream was served before the session began and the presentations offered interesting ideas, generated lively discussion, and were well attended.

Epoch Parallelism: One Execution Is Not Enough

Jessica Ouyang, Kaushik Veeraraghavan, Dongyoon Lee, Peter M. Chen, Jason Flinn, and Satish Narayanasamy, University of Michigan

Jessica Ouyang presented a new style of parallelism called "Epoch Parallelism," a novel approach to writing multi-threaded programs. Since it is hard to write multi-threaded programs that are fast and correct, Ouyang proposed that programmers could write two programs: one which was fast but buggy and another which was slower and correct. Programs are then split into multiple epochs and the output from the first epoch of the faster program can be used to accelerate the execution of the following epochs in parallel. If a checkpoint from the faster program does not match that from the slower execution, the speculatively executed parts of the program are rolled back.

Michael Vrable from UCSD asked how this execution pattern could be used with I/O or network interactions. Ouyang agreed that output can be externalized to the user only when the slower process completed, but that different parts of the application could internally proceed using the speculative results. Emin Gün Sirer from Cornell inquired about applications which could make use of the correctness-speed tradeoff. Ouyang clarified that the faster executions in this model were not entirely buggy and would mostly give the correct answer. Examples for this included avoiding additional runtime assertion checks and optimistic concurrency techniques such as lock elision.

Automated Software Reliability Services: Using Reliability Tools Should Be as Easy as Webmail

George Candea, Stefan Bucur, Vitaly Chipounov, Vova Kuznetsov, and Cristian Zamfir, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

The second research vision proposed the creation of a software certification service and was presented by George Candea (EPFL). Although many tools are being developed to help test, debug, and verify correctness of software, Candea observed that these were not having much of an impact in the real world. To make software reliability more effective, Candea argued that reliable software should have an advantage over competitors in the market.

He then presented a case study of how Underwriter Labs had established a safety standard for electrical appliances early in the twentieth century and how a similar certification service could promote software reliability. This service would use automated techniques to objectively measure the reliabil-

ity of binaries and would provide ratings similar to crash-test ratings for cars. Candea listed the systems research problems related to building such a service. These included automated proof generation techniques and scalable testing of binaries on massive clusters.

Peter Chen, University of Michigan, pointed out that there were blogs which provided such ratings for antivirus software today and that software was tuned to perform well on these benchmarks. Candea replied that the certification service would be based on concepts like model checking and symbolic execution, which would make tuning the software more difficult when compared to benchmarks. Ivan Beschastnikh, University of Washington, asked if the certification service was applicable to all kinds of software. He noted that comprehensive testing was already used in critical software in embedded systems and that users might not care much about the reliability of services like Facebook. Candea replied that today, users chose software based on functionality and that it could be possible that they would continue to do the same. However, he hoped that presenting data about reliability of software would help them make a better decision.

SIGOPS to SIGARCH: "Now it's our turn to push you around"

Jeffrey C. Mogul, HP Labs

Jeff Mogul from HP Labs presented his thoughts on how operating systems were being ignored by the computer architecture community and what the SIGOPS community could do to change this. Mogul argued that OS designers used commodity hardware to build their systems and had stopped asking the architects for newer features like core-to-core message passing primitives or interfaces for managing thermal issues.

Analyzing the problem in detail, Mogul pointed out that architects had no open-source simulator which could run real operating systems, and the OS community could help articulate and build such a simulator. Also, there was a need for operating system benchmarks which could be run by architects in simulators to evaluate how their changes affect the OS. Finally, Jeff also argued that SIGARCH should be encouraged to accept papers which focus on OS issues in conferences like ASPLOS.

This was followed by a lively question-answer session. Margo Seltzer from Harvard disagreed with the presentation and argued that the best compliment for an OS designer was for the user to not realize that the OS was running. Mogul replied that it would be some time before we had a zero-overhead operating system and that we need better cooperation between the hardware and the OS community for that to happen. David Lillethun from Georgia Tech asked how the inertia about changing instructions sets, especially x86 for desktops, affected some of the problems pointed out earlier. Mogul jokingly remarked that his terminal still runs on an Alpha computer and that he felt managing the memory system was a greater challenge than the instruction set. Rik Farrow argued that CPU architecture features, such as the trap into privileged mode, actually shape how operating systems work, noting that there was a paper in this conference about avoiding the penalties for making that context shift. Farrow applauded the idea that the systems and architectures groups would actually communicate, instead of systems researchers just working around problems caused by architecture. Emin Gün Sirer's comment, that he could imagine a corresponding talk at WWW about how the OS community had not provided the database community with the right interface and had forced them to write to raw disk, drew laughter and applause from the audience.

Embrace Your Inner Virus

Michael F. Nowlan and Bryan Ford, Yale University

The final research vision presentation at OSDI '10 was by Michael Nowlan from Yale University, who proposed a system design in which the OS expects applications to behave like viruses. First, Nowlan noted that viruses were prevalent and transferred easily across machines through the Internet and other media like Bluetooth and USB. Instead of trying to fight viruses, he proposed that the OS should be designed to handle viral applications (vapps) using techniques like code sandboxing and information flow control.

Nowlan presented several case studies of how vapps could be useful. The first case study was about a contact application which spread to all the users in a particular room. This could be useful for users to get in touch with other attendees at a conference. The second case study analyzed a photo uploading vapp that could be used to disseminate photographs to other machines as soon as they are taken. If the original machine is lost, users can recover their photos by being "re-infected" from other machines.

Nowlan also discussed various business models that could be used to build such viral applications. Most interesting among these was the "Quid Pro Quo" model where the user could consume a service for free but had to submit a local resource in exchange. For example, a weather application on a phone could be used for free but the user would need to upload the local temperature from the phone's thermometer. Following the presentation, Nowlan was asked if the bad vapps would not affect the system. He replied that they were proposing greater transparency into applications and the use of techniques like information flow control in OS design which could overcome such problems.

Deterministic Parallelism

Summarized by Alan Dunn (adunn@cs.utexas.edu)

Deterministic Process Groups in dOS

Tom Bergan, Nicholas Hunt, Luis Ceze, and Steven D. Gribble, University of Washington

Tom Bergan presented dOS, a set of modifications to the Linux kernel to allow groups of processes to run deterministically. Nondeterminism in OS execution makes bugs harder to reproduce and causes replicas in state machine replication-based fault-tolerance protocols to become unsynchronized. To avoid these problems, dOS introduces the ability to create a "deterministic box," in which contained processes execute in a deterministic manner. Bergan noted that dOS guarantees a deterministic ordering of internal operations (including inter-thread communication), but does not guarantee deterministic performance.

Deterministic execution is not a new problem, but the dOS project improves over prior work both conceptually and in practice. Bergan explained that dOS incorporates a distinction between "internal" and "external" nondeterminism. The difference is that internal nondeterminism is controllable by the operating system, while external nondeterminism is caused by uncontrolled sources such as user input and network packets. Internal nondeterminism can then be controlled by use of algorithms that ensure deterministic outputs, such as deterministic ordering of IPC operations and process scheduling. Only external nondeterminism needs to be logged and replayed to processes. Bergan pointed to log sizes that were several orders of magnitude smaller than a prior full-system determinism system as evidence for the utility of the selective determinism approach.

The main pieces of dOS's implementation are the DMP-O deterministic scheduler and a "shim" for monitoring external inputs and controlling their delivery. While DMP-O is from prior work, dOS still needed a way to detect communication between threads, which was accomplished by modifications to system calls and page protections for shared memory. Threads can then be run in parallel until communication occurs and communication events are serialized in a deterministic order. Discussion of the shim included two examples: deterministic record and replay, and replication of a multi-threaded server.

Questions for the presentation ranged over topics from technical comments on performance to high-level questions about design motivation. Amittai Aviram (Yale University) and Micah Brodsky (MIT) asked whether performance issues incurred by fine-grained shared memory between processes could be resolved. Bergan commented that previous systems had used specialized hardware or compiler-inserted instrumentation to solve this problem, and that still other approaches might be possible. Madan Musuvathi of Microsoft Research wondered whether nondeterminism was such a bad thing and whether there might be tradeoffs between degrees of nondeterminism and performance. This was a key theme for this session, as the three different systems presented managed this tradeoff in different ways. Bergan said that the notions of internal and external nondeterminism were dOS's way of handling this: in dOS, one can select the processes that need to be deterministic, and thus influence what portions of nondeterminism are internal and external.

Efficient System-Enforced Deterministic Parallelism

Amittai Aviram, Shu-Chun Weng, Sen Hu, and Bryan Ford, Yale University

- *Awarded Jay Lepreau Best Paper!*

Bryan Ford presented Determinator, a microkernel designed to eliminate data races, which was awarded one of two Jay Lepreau Best Paper awards for OSDI '10. There are currently many systems devoted to helping programmers manage existing data races that occur in conventional operating systems. Ford said that despite all of this, it would be nice if programmers didn't have to worry about data races and were not forced into using specific programming languages to achieve this end. Determinator is an attempt to accomplish this: it is a microkernel with a small set of system calls to manage shared state. Facilities that would be present in modern monolithic UNIX systems, such as the C library, process management, and file system API, are instead implemented in user space and take advantage of these system calls to eliminate data races.

The microkernel system calls are built on the paradigm of "check-out/check-in" of state. When a kernel thread forks, it creates a local copy of its parent's address space. Any reads and writes that this thread performs are then against its local copy. When the thread joins its parent, its differences from the parent are merged back to the parent's state. Ford described how this solves data race issues: read-write races are eliminated, because writes will occur to a different local state than reads, while write-write races are detected upon join and can be trapped and then resolved.

Determinator was evaluated by comparing its speed on benchmark programs to that of Linux. The primary factor influencing benchmark speed was the granularity at which the changes children made to shared state were found and merged back into a parent, with finer granularity leading to worse performance.

There were a number of interesting questions after this presentation. An audience member from Microsoft Research asked about the relationship between Determinator's approach to shared state and transactions. Ford responded that the two have similar isolation properties, but that transactions generally do not provide determinism. Eric Eide of the University of Utah asked about whether it might be better to mandate use of a specific language for Determinator. Ford said that while a single language might help by allowing easier introduction of primitives to manage granularity of shared state, it seemed advantageous to allow application programmers greater language freedom. Jason Flinn of the University of Michigan commented that experience with systems like CODA showed that resolving write-write conflicts can be challenging, and he wondered how difficult this was with Determinator. Ford said that his team did not have enough experience writing applications for Determinator to comment definitively, but that he believed there would be some issues to explore with respect to when changes to state are propagated.

Stable Deterministic Multithreading through Schedule Memoization

Heming Cui, Jingyue Wu, Chia-che Tsai, and Junfeng Yang, Columbia University

Heming Cui presented Tern, a system for improving system determinism via capturing and reusing thread schedules. Tern shares some of the motivation of other papers from the Deterministic Parallelism session in that it also aims to allow for easier testing and debugging through increased determinism. However, Tern also targets a difficulty found in other systems: often a small change in program inputs causes radical changes in the schedules produced, which eliminates some of the benefits of deterministic scheduling. Cui described how Tern avoids this by representing schedules as sequences of synchronization operations, which can potentially be reused for multiple inputs. It calculates sets of constraints that must be satisfied by an input to use a certain schedule and caches schedules with their constraints.

Cui illustrated the use of Tern with example code from PBZip2. To use code with Tern, it is necessary to first annotate the variables that are important to scheduling. The code is then instrumented at compile time to allow Tern to record and control scheduling. For any given program run, constraints on the annotated variables are created based on the branches that the program takes. These constraints are stored with the resultant schedule in a memoizer. The constraints will be evaluated for subsequent program runs, and the stored schedules will be used if their constraints are satisfied.

Tern was evaluated based on its degree of schedule reuse, stability in bug reproduction, and overhead. For several real-world programs (Apache and MySQL), Tern was able to use 100 or fewer schedules to cover significant stretches of execution, corresponding to a 90% schedule reuse rate. Computational overhead was often less than 10%.

Ding Yuan of the University of Illinois at Urbana-Champaign claimed that Tern would not eliminate data races. Cui pointed out that this is true, although this was a tradeoff made to allow schedule reuse, thus increasing performance and stability. An audience member asked about whether constraint size and number could affect performance, and Cui said that they have techniques for removing redundant constraints and eliminating irrelevant constraints to mitigate this issue. Bryan Ford of Yale asked whether Tern memoizes schedules only at whole program granularity, and whether other granularities would be useful. Cui responded that currently only whole program granularity is supported, but that others could be useful.

Systems Management

Summarized by Don Porter (porterde@cs.utexas.edu)

Enabling Configuration-Independent Automation by Non-Expert Users

Nate Kushman and Dina Katabi, Massachusetts Institute of Technology

Nate Kushman presented KarDo, a tool for automating configuration and other administrative tasks. The key problem addressed is that users generally know what they want, but do not know how to do it. There is no easy way to automate fairly common tasks that require GUI manipulation. The KarDo system improves on the state of the art by collecting a few traces (generally two) of experts performing a given task, and then distilling a generalized trace which can be replayed on an end user's computer. KarDo requires no kernel or application modifications. Kushman presented details of how they generalize their traces. KarDo uses a support vector machine to classify GUI actions into three categories: update, commit, and navigate. KarDo uniquely identifies GUI widgets by their text, which is extracted from the accessibility interface. In generalizing a trace, navigation is separated from the rest of the trace; using all traces, the needed navigation steps are generalized for a canonical trace. Kushman also presented algorithms for removing needless actions and handling differences in the GUI.

KarDo was evaluated using 57 tasks taken from the Microsoft Help and eHow Web sites. The authors collected traces from a set of diversely configured virtual machines and then tested them on a different set of virtual machines. Existing

automation techniques worked on only 18% of the test virtual machines, whereas KarDo had an 84% success rate. Of the 16% of tests that failed, 4% were navigation errors, 5% had missing steps, and 7% were classifier errors. More traces should lower these error rates.

Adam Chlipala of Harvard asked why they didn't just parse configuration files, and Kushman said that they could go quite a bit farther than just parsing files. Someone asked whether a collected trace could reveal private information. Kushman responded that the traces used for training can include private data, but if any two traces have different values, these are considered private and filtered out of the canonical trace. Josh Triplett from Portland State University asked whether the same tool could be applied to command-line tasks. Kushman responded that a key challenge is text processing, which would require additional insights to produce canonical traces of the same quality.

Automating Configuration Troubleshooting with Dynamic Information Flow Analysis

Mona Attariyan and Jason Flinn, University of Michigan

Mona Attariyan presented ConfAid, a tool that automatically identifies the root cause of a configuration error. The motivating problem is that configuring software is difficult and prone to user error. Currently, if a user can't get a piece of software to work, s/he must ask colleagues, search a manual, or look at the code if it is available. Users need better tools to troubleshoot these errors. The usage model of ConfAid is that a user runs the misconfigured software under ConfAid and it outputs a list of likely root causes.

ConfAid works by tracking dynamic information flow through the application binaries. ConfAid does not require source code or debugging symbols. Attariyan described several technical details of how they use information flow analysis of control and data flows to assess which configuration variables influenced the program control flow that led to the incorrect behavior. In this analysis, there is a fundamental tradeoff between the precision of the analysis and performance; ConfAid uses imprecise analysis and introduces three heuristics to improve performance and lower the false-positive rate.

To evaluate ConfAid, the authors tested 18 real-world errors from OpenSSH, Apache, and PostFix (collected from manuals, support forums, etc.), and 60 randomly generated errors. For the real-world errors, ConfAid ranked the correct root cause first for 72% of the errors and second for the rest. Among the random errors, ConfAid ranked the correct root cause first or second for 55/60 errors. The average execution time of ConfAid was 1 minute, 32 seconds.

Someone asked whether ConfAid required an understanding of configuration file syntax, Attariyan responded no. Alice Zheng of Microsoft Research asked if all bugs the authors dealt with were single-mistake bugs, and whether they expected all bugs in the real world to be single-mistake. The authors only looked at single-mistake bugs. ConfAid would likely suggest both mistakes independently, but cannot output that both configuration variables must be changed in tandem. Regarding real-world configuration errors, it is hard to say, but the test errors they used seemed quite common on the forums.

Lunchtime Award Announcements

Summarized by Rik Farrow (rik@usenix.org)

Two awards were announced during the symposium luncheon. Robert Morris of MIT received the Mark Weiser award for an individual who has demonstrated creativity and innovation in operating systems research. Roger Needham and Michael Schroeder received the SIGOPS Hall of Fame Award for their 1978 paper on authentication over a non-trusted link (Needham-Schroeder protocol). Schroeder was present to accept the award.

Inside the Data Center, 2

Summarized by Don Porter (porterde@cs.utexas.edu)

Large-scale Incremental Processing Using Distributed Transactions and Notifications

Daniel Peng and Frank Dabek, Google, Inc.

Frank Dabek described the new Percolator indexing system used at Google to reduce the delay between Web crawling and re-indexing. Prior to Percolator, there was a delay of days between crawling a new document and its incorporation into the index, as the entire corpus of crawled documents was reindexed. The solution described is incremental reindexing, which introduces a number of new challenges, described in the talk. In order to support incremental indexing, the authors added distributed transactions to BigTable, which provides snapshot isolation semantics, and added notification support.

Among the challenges described was the problem of "Bus Clumping," the problem that the randomized scans for new work tend to clump working on data that is time-consuming to process, reducing parallelism and overloading servers. The solution they adopt is trying to acquire a lightweight scanner lock per row of BigTable. If the lock acquire fails, the scanner jumps to a random point in the table to look for new work—the equivalent of a city bus teleporting ahead in

the route. Percolator also stressed certain new errors in the Google cluster, including a set of failing CPUs that randomly failed to XOR bits correctly, and an incorrect resistor value that powered off a certain motherboard. Dabek concluded with advice based on this experience: (1) push performance debugging through all layers of the system and (2) expect weirdness proportional to machine count.

The talk concluded with the assertion that Percolator is an existence proof that distributed transactions can be implemented at Web scale. This was reflected in a question from David Cock of the University of New South Wales regarding the novelty of the work; Dabek answered that the novelty is the scale of the system, which the field had given up on. Mehul Shah of HP Labs asked about the limits of the system and how it handled stale locks left by clients. Dabek responded that the largest problem with concurrency was heavy write conflicts, which were addressed with a backoff heuristic. The space required to store notifications in BigTable is not an issue, and stale locks were cleaned up lazily. Margo Seltzer of Harvard University asked his thoughts on the debate between MapReduce versus databases. Dabek said that if you pretend MapReduce is a database, it is a bad one, but that MapReduce is not dead and is still used heavily within Google.

Reining in the Outliers in Map-Reduce Clusters using Mantri

Ganesh Ananthanarayanan, Microsoft Research and UC Berkeley; Srikanth Kandula and Albert Greenberg, Microsoft Research; Ion Stoica, UC Berkeley; Yi Lu, Microsoft Research; Bikas Saha and Edward Harris, Microsoft Bing

Srikanth Kandula presented the Mantri system, used in production at Microsoft Bing. The goal of Mantri is reducing outliers, which slow down MapReduce jobs; reducing outliers improves productivity, gives cloud service providers more predictable completion times, and better utilizes datacenter resources, saving money.

One key cause of outliers is unavailable inputs at a computation node. To address this, Mantri replicates intermediate data and introduces heuristics to predict what data to replicate where, weighted by the cost of recomputation. A second key cause of outliers is variable network congestion, which Mantri addresses by carefully placing reduce tasks such that traffic on a link out of a rack is proportional to the bandwidth. Although global coordination to load-balance the network links across all jobs is difficult, having each job balance its own traffic is a good approximation of the ideal. A final cause of outliers is workload imbalance, often due to contention at a given machine. There is a long tail (25%) of miscellaneous causes for this.

Based on experience working with Mantri, Kandula recommended three principles for managing MapReduce systems: (1) predict to act early, (2) be aware of resource and opportunity cost of an action, and (3) act based on the cause. The overall result of deployment in the Bing cluster is a median 32% reduction in job completion time and lower utilization.

Someone asked what Kandula would do if they had a non-uniform cluster, say, with some machines that were faster but handled fewer requests. Kandula answered that they have a scheduling system that normalizes machine capabilities to a slot abstraction, which are scheduled rather than entire machines. Emin Gün Sirer from Cornell asked how the authors might change the MapReduce interface to address stragglers, given what they know from experience. Kandula responded that they would have an interface to yield more even partitions than simple hashing.

Transactional Consistency and Automatic Management in an Application Data Cache

Dan R.K. Ports, Austin T. Clements, Irene Zhang, Samuel Madden, and Barbara Liskov, MIT CSAIL

Dan Ports presented a system called TxCache, which provides transactional consistency for an in-memory database cache such as memcached. The key problem this work addresses is that modern Web applications face immense scaling challenges, but existing caching techniques only offer limited help. For instance, personalization on sites like Facebook foils whole-page caching. Similarly, database caches are of limited use, since Web applications require increased post-processing of data in the application itself. Application layer caches, such as memcached, provide a useful alternative. By caching application objects, these caches can separate common and customized content and reduce overall server load. The key challenge to this approach is that current application-level caches do not provide transactional consistency, leaving the application to address transient anomalies.

Ports then described the TxCache system, which provides a simple interface to delineate transactions on cacheable data. TxCache provides bounded staleness for transactions, allowing read-only operations to improve performance by returning slightly stale data where safe. Programmers can also specify cacheable, side effect–free functions, allowing the system to cache their results and avoid needless recomputation. Ports then described several key challenges, including selection of timestamp intervals and maintaining coherence through invalidations.

The system was evaluated using the RUBiS benchmark with a single database server and nine front-end/cache servers. The experiments showed that a larger cache yielded a higher

hit rate and better performance. Allowing stale results also improves performance by as much as 3–5x, but the knee of the performance curve was around 20–30 seconds. The authors also measured the overhead of consistent caching by allowing inconsistent reads; performance improved only marginally, indicating that the costs are negligible.

Marcos Aguilera of Microsoft Research asked whether TxCache provided serializability or snapshot isolation. Ports answered that the system provides either, dictated by the guarantees of the underlying database. Ports was also asked whether they got performance wins on a single system, or only across several nodes. He answered that the number of nodes determines the size of the cache and the hit frequency.

Piccolo: Building Fast, Distributed Programs with Partitioned Tables

Russell Power and Jinyang Li, New York University

Russell Power presented Piccolo, a framework for developing high-performance distributed applications. Problems that can fit into memory in a cluster are a key motivation for this work. Power structured the talk around page rank as a representative example. Existing data flow models, such as MapReduce, don't expose global state, and models such as MPI and RPC require explicit communication, making them harder to write. The goal of Piccolo is to provide distributed, in-memory state. The runtime system transparently handles communication when the programmer requests reads and writes.

Power described several technical challenges in developing a page rank example application on Piccolo and challenges in implementing Piccolo on a cluster. Throughout the talk, he refined a pseudocode example that was both reasonably detailed and simple enough to fit onto one slide. For instance, Piccolo must add synchronization primitives for concurrent writes to a variable. Because many writes are actually accumulator functions, these can be used with release consistency to improve concurrency. Power also explained that storing state at nodes increases the complexity of load balancing, as it is harder to start new jobs. Starting jobs at remote nodes gives up locality and harms performance; moving is hard because the old site must still forward updates to the new site.

The system was evaluated on a cluster of 12 nodes totaling 64 cores. Compared to Hadoop, calculating page rank on a 100M page baseline, Piccolo was substantially faster. The iteration time remained nearly flat as more workers were added to match a larger input graph, indicating near-perfect scaling. The Piccolo code is available at piccolo.news.cs.nyu.edu.

Josh Triplett asked what fall-back strategy they used for non-commutative aggregate functions. Power answered that these were relatively rare in their experience and that they could use pairwise locking, but that locking was slow enough to avoid at all costs. Daniel Greenway asked about checkpointing and what they did if nodes fail. Power replied that they roll all nodes back to the last checkpoint.

Cloud Storage

Summarized by Katelin Bailey (katelin@cs.washington.edu)

Depot: Cloud Storage with Minimal Trust

Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish, The University of Texas at Austin

Prince Mahajan presented Depot, an attempt to remove trust from cloud storage. This system was unable to completely remove trust from the equation, but Mahajan argued that it comes very close: for *put* availability, consistency and staleness detection, the system requires no trust, while it minimizes the trust necessary for *get* availability and durability (reliant on one node).

Depot has multiple failover servers and can default to client-to-client communication in the case of errors. As with other work in the area, Depot uses both local state and metadata added to commits to allow clients to check the history and detect forks in the state. The system uses Fork-Join-Causal consistency in the case of unreliable nodes, which allows for taking a forked subset and reconciling it as if it were two concurrent commits from "virtual nodes." It also allows for the eviction of a node that is consistently faulty or malicious. Mahajan then covered implementation details and performance evaluation for the Depot project and the related "teapot" project implemented on Amazon's S3, demonstrating almost unmodified server-side code. He claimed that the performance overhead was modest and the implementation practical for use.

There were a large number of questions following the talk. Dave Koch of NICTA pointed out that CPU overheads on clients were high. Mahajan conceded that one of the graphs showed a 400% CPU overhead for one test, but argued that CPU cycles are cheap enough to allow this overhead to be modest nonetheless: throughput is the concern. David Schultz pondered the correctness of clients during reads: for example, a Byzantine client who had the only copy of some data. Mahajan clarified that individual clients can have replication specifications or filters, such as only reading data that is resident on four nodes. This would, however, reduce availability when only one node is online, as Schultz pointed out.

Comet: An Active Distributed Key-Value Store

Roxana Geambasu, Amit A. Levy, Tadayoshi Kohno, Arvind Krishnamurthy, and Henry M. Levy, University of Washington

Roxana Geambasu presented Comet, a variation on distributed hash tables (DHT), which allows applications to make use of small bits of code inserted in the DHT. Motivated by earlier work (Vanish, Geambasu 2009) which exposed the frustrating nature of working with a one-size-fits-all DHT serving many applications, the project produced a system that is flexible, lightweight, and provides isolation of the included code. The goal was to create an extensible DHT that can offer different capabilities to the different applications.

The implementation of this project focused on a unit called the Active Storage Object (ASO), which consists of data and tiny snippets of code, written in a very restricted form of Lua. The architecture of the system consists of an active runtime over the standard DHT, which ASOs can access via handlers and an ASO API. The ASO is sandboxed and has a very limited interface to the outside world, enhancing the overall security of the system. The sandbox allows the ASO to have some knowledge of the host and the DHT in general, but restricts actions to periodic tasks, interceptions of accesses, and minimal DHT actions (put, get, lookup). Geambasu demonstrated how even these limited ASOs can create powerful capabilities for the application. Her examples include an altered replication policy, a proximity tracker for closest peers, and self-monitoring code. She pointed out that Comet not only allows policies to differ between applications, but opens up new opportunities for tracking and analyzing usage, or even adding debugging or logging possibilities.

Pietros Maniatos of Intel Labs, Berkeley, wondered about security. Geambasu noted that there were global restrictions set on the ASOs such that they could not take over the host, and clarified that the only classes of operations allowed in the modified Lua code were math operations, string operations, and table manipulation. Additionally, there is a hard limit on lines of code allowed. She also addressed the question of replicas sharing data or being treated separately: the replicas are indeed treated as separate copies, but they can locate and even merge with each other, if desired. Dutch Meyer asked about variance in performance, for example, with a garbage collector, and Geambasu suggested they might use another type of language. Ben Wester (U. Michigan, Ann Arbor) asked if code gets run on every get, and Geambasu replied that in a traditional DHT, things may fail, requests get dropped, so some redundancy is a good thing.

SPORC: Group Collaboration using Untrusted Cloud Resources

Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten, Princeton University

Ariel Feldman presented a system for building collaborative projects on an untrusted server, motivated by the desire to use cloud services without trusting the provider. The system presented a single server with a number of clients, moving the application state to the client, as well as a copy of state stored client-side and all server storage being encrypted. The system relies on the previously researched ideas of fork* consistency and operational transformation (OT).

Feldman took the audience through a number of common scenarios. He outlined how the fork* consistency is represented by a history hash embedded in each commit, including sequence numbers and corresponding checks when an update is pushed out to the clients. In addition, Feldman demonstrated in detail how the OT transform functions can be used to deal with merging forked groups, as well as handling pending transactions and offline commits. Lastly, Feldman talked about how access control works in SPORC: symmetric keys (for efficiency) keep the commits encrypted in transit and on the server. Access control list (ACL) changes are performed with barriers to prevent conflicting changes; all changes after a barrier are rejected until the change is fully committed. Redistribution of AES keys is done via encryption as well, preserving a chain of encrypted keys for use by members joining later. Feldman argues that at no point would the server have to be trusted. Performance evaluations on a local-area network indicate that SPORC would have usable latency and throughput for moderate-sized groups.

Bryan Ford of Yale proposed a scenario where Alice attempts to evict (malicious) Bob from the access list, but Bob adds (equally malicious) Eve and Fred before he is evicted, and the process cascades such that there are more and more malicious members of the community. Feldman explained that there are three levels of access possible in SPORC: administrators, editors, and readers, providing some assurance, although a rogue administrator could cause substantial problems. Josh Triplett probed the choice to use servers, if they are simply used for ordering and storage. Feldman replied that they allow for a more timely commit process than a decentralized system. He also verified that the project assumed a correct client. However, he pointed out that fork* recovery allows for an undo option, if necessary.

Production Networks

Summarized by Peter Bailis (pbailis@eecs.harvard.edu)

Onix: A Distributed Control Platform for Large-scale Production Networks

Teemu Koponen, Martin Casado, Natasha Gude, and Jeremy Stribling, Nicira Networks; Leon Poutievski, Min Zhu, and Rajiv Ramanathan, Google; Yuichiro Iwata, Hiroaki Inoue, and Takayuki Hama, NEC; Scott Shenker, International Computer Science Institute (ICSI) and UC Berkeley

Jeremy Stribling presented Onix, a software system for controlling large-scale networks. Onix is a cross-institution effort designed for real-world production network deployments and is currently under quality-assurance testing. Jeremy began by describing typical router architecture: a fast-path forwarding plane determines where to send packets, while a control plane inside the device can reprogram the forwarding plane and handle exceptions. Centralizing the control plane as in Software-Defined Networking (SDN) allows greater control over the network, but many of the issues related to scalable rule propagation and device reprogramming are difficult. Onix moves this centralized control logic to a distributed infrastructure while providing a high-level interface, solving state distribution problems and abstracting low-level mechanisms. Onix has several goals, including generality, scalability, reliability, simplicity, and performance.

For generality, developers program against a network graph called the Network Information Base (NIB) in which nodes are physical entities like switches, hosts, and ports. The NIB is the focal point of the system, and Onix takes care of talking to other instances, importing external state changes and exporting local state changes. Because different data has different storage requirements, it can be stored in replicated transactional storage (SQL) or in a one-hop in-memory DHT. For an ARP server, for example, the switch topology would be stored as hard state in the transactional storage, and the IP and MAC soft state would be stored in the DHT. This storage is specified pre-runtime, and at runtime the programmer only interacts with the NIB.

Scalability and reliability concerns amount to distributed state management problems. There are application-specific tradeoffs involved in determining how to partition the network. For example, in some networks Onix need not connect to every switch, which is more scalable than connecting to all of them. Traditional network scaling uses partitioning (VLAN systems) and aggregation, which reduces fidelity. Onix instead uses different instances to control different parts of the network, and network equipment connects only to a subset of instances. Onix can also aggregate data, reducing its fidelity before sharing. For example, when exporting uptime data, Onix can present an average of the information to other nodes instead of providing an exhaustive dataset. Link failures are the application's responsibility, but Onix assumes a reliable management connectivity or uses a multi-pathing protocol. Failures are handled using distributed coordination (via Zookeeper integration). Performance details are available in the paper.

Jeremy stressed that Onix is a real, production program. A prototype of over 150,000 lines of code is currently under testing and expected to be deployed in future products in the next few months or within the year.

Marco Canini from EPFL asked several questions about the application of Onix. Jeremy explained that there is only one kind of routing protocol per network, and that the Onix "application" is a program that manipulates the NIB graph to set up the routing the way that the application developer requires. There are several protocols for exchanging messages, but one easy way is using the OpenFlow protocol. Network bootstrapping depends on the particular configuration. An attendee from UCSD asked how the Onix ARP cache could be refreshed. Jeremy said that hosts sent out ARPs periodically. Eddie Kohler from UCLA asked about Jeremy's favorite implementation trick in Onix. Jeremy likes the fact that Onix provides a lot of the distributed mechanisms that are hidden from the programmer, as well as the fact that distributed storage can be swapped out transparently to the programmer.

Can the Production Network Be the Testbed?

Rob Sherwood, Deutsche Telekom Inc. R&D Lab; Glen Gibb and Kok-Kiong Yap, Stanford University; Guido Appenzeller, Big Switch Networks; Martin Casado, Nicira Networks; Nick McKeown and Guru Parulkar, Stanford University

Rob Sherwood described a new technique and prototype implementation called FlowVisor which allows realistic evaluation of new networking services within production networks. It's difficult to evaluate new services in practice. Services may not be fully ready for production, but there's a need to test them. Rob described real networking as a black-box system, with hundreds of thousands of devices, while no one really knows realistic Internet topologies. Some testbeds use software routers, but performance can suffer due to limited hardware scale, artificial topologies, and limited adoption, leading to the use of synthetic data. Subsequently, the driving motivation for this technique is to allow production networks to serve as testbeds while providing strong isolation between production and testbed networking. Rob believes that the production network can indeed be the testbed.

Rob described *network slicing*, a new technique for accomplishing this isolation. The network is partitioned into logical copies called slices, each of which controls its own packet forwarding. Users pick which slice controls their traffic, and existing production services and testbed services can run in separate slices. This enforces isolation, while allowing the logical testbed to mirror the production network topology. Network slicing can be accomplished by multiplexing the data plane, a custom, high-performance ASIC within the router that enforces rules between multiple control planes, which compute forwarding rules and push them to the data plane. The data plane is unmodified, allowing forwarding without performance penalty, while the multiple slices share the general-purpose CPU on the router.

Rob described FlowSpace and FlowVisor, an implementation of network slicing. FlowSpace maps packets to different network slices according to twelve OpenFlow header types, including IP address, MAC address, and TCP port. This allows users to opt into slices at a fine granularity, like HTTP, VoIP, or other network services. FlowVisor controls the control plane using the OpenFlow protocol, allowing external control. FlowVisor handles exceptions from the data plane and forwards them to the slice controller, which checks the policy and forwards it to the router. Handlers are cached for scalability. To keep slices from monopolizing the CPU, the system currently rate-limits rule insertions and uses periodic drop-rules to throttle exceptions, although future systems should have proper rate limiters. FlowVisor is currently deployed on or will be deployed on eight campuses and with two ISPs.

Jeff Mogul from HP Labs noted that the CPU is a limited resource and that this might limit the flow set-up rate, which Rob acknowledged. An attendee from UBC asked about the effect on latency. Rob showed that the average latency is approximately half a millisecond but depends on the operation. Another attendee asked about the requirements for forwarding memory. Rob agreed that the number of rules is a scarce commodity on current routers, but hardware manufacturers are working on expanding this. Rob mentioned that there are several techniques (e.g., caching) for operating without a complete set of forwarding memory. Finally, Josh Triplett from Portland State University asked whether the authors had considered using some slices as control slices for the system. Rob responded that in practice the experimental slices use the production slice for the control plane.

Building Extensible Networks with Rule-Based Forwarding

Lucian Popa, University of California, Berkeley, and ICSI, Berkeley; Norbert Egi, Lancaster University; Sylvia Ratnasamy, Intel Labs, Berkeley; Ion Stoica, University of California, Berkeley

Lucian Popa presented rule-based forwarding (RBF), a technique for allowing flexible Internet packet forwarding. The goal of this work was to allow more general forwarding directions, which provide routers with information on how to send their packets. This generality could provide the ability to use middleboxes, source routes, or in-network packet processing. Their thesis is that flexibility needs to be balanced by providing policy-based access, and there is a balance between flexibility and constrained access. The idea here is that forwarding directives are carried inside packets; routers only need to verify that the packet complies with the policies of all involved parties and forward the packet. Lucian argued that this allows the appropriate balance between flexibility and policy enforcement.

Lucian presented a rule-based forwarding architecture. Rules are leased from and certified by trusted third parties, and all entities involved in the rule are certified. The RBF control plane consists of both a distribution infrastructure and a certification infrastructure. RBF assumes an anti-spoofing mechanism, the existence of rule-certifying entities, and a DDoS-resistant rule distribution. Rules are represented as a sequence of if-then-else statements that are comparison operations on router state, along with several actions (e.g., drop packet); however, rules cannot modify router attributes. Thus, the rules are flexible (allowing many policies), compliant (certified), and safe (cannot modify state). In practice, rule forwarding incurs little size overhead (between 60 and 140 bytes, or 13% on standard IP packet, 27% with RSA signatures) and limited runtime overhead. They incur negligible overhead on a software router on a fast path and a 10% slowdown on the slow path, when verifying RSA signatures and using real traffic.

Michael Walfish from UT Austin asked about the feasibility of determining if a particular inter-domain path was taken. Lucian answered that cryptographic guarantees can solve this problem. Michael also asked about unnamed stakeholders in the network—how can we name rule entities in the network a priori? Lucian claimed that if we treat rule-based forwarding like an overlay network, this is not a problem; after considerable discussion, this question was taken offline. Andreas from AT&T Research asked about the effect on latency, which is negligible in the current implementation. Helen Wang from Microsoft Research asked about the deployability on today's ISPs. Lucian answered that a partial deployment is possible and might be able to detect whether

some packets are or are not policy-compliant, but some packets would still be received. Rule-based forwarding enables more services, Lucian claimed, so this is still a benefit for ISPs.

Mobility

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

TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones

William Enck, The Pennsylvania State University; Peter Gilbert, Duke University; Byung-gon Chun, Intel Labs; Landon P. Cox, Duke University; Jaeyeon Jung, Intel Labs; Patrick McDaniel, The Pennsylvania State University; Anmol N. Sheth, Intel Labs

William Enck explained how to balance fun and utility with privacy. He detailed how applications downloaded to a smartphone have full access to potentially private information, such as the GPS unit, microphone, camera, address book, and the phone's unique ID. Too often, users are surprised at what information is transmitted from the phone. In one such example, a popular wallpaper application was sending users' phone numbers back to the developer. While this particular example was found to be non-malicious, it serves as a warning that our software may be revealing information that users are not comfortable disclosing.

TaintDroid is a VM-based taint tracking system for Android phones. In order to determine when private information has left the phone, the authors modified the Dalvik interpreter to store and propagate taint tags for variables. This allows private data to be tracked, for example, from its source in the address book until it is released to the network. Despite operating in a resource-constrained environment, TaintDroid runs in real time. It displays warnings during live use of an application and was shown to have only a 14% overhead. The authors evaluated the system on 30 applications randomly selected from the 50 most popular applications in the Android Marketplace. Ultimately, 105 connections transmitted private data, while only 37 of those messages were clearly legitimate. Of the 30 studied applications, 15 shared location information with advertisement servers, and 7 shared device identifiers with a remote server. In no cases were these disclosures described in the EULA or evident from the user interface.

Iqbal Mohomed from Microsoft Research asked about implicit information flows, such as using timing perturbations to leak information to another process. Enck acknowledged that this is an existing problem, one that can be addressed with static analysis, although doing so introduces trade-offs such as high false-positive rates. Volodymyr Kuznetsv from EPFL exclaimed that the work almost made him throw away his mobile device! He encouraged the authors to provide the tool as a Web site that can test applications uploaded by concerned users. In answering other questions, Enck clarified that TaintDroid identifies when private data has left the phone, as opposed to detecting privacy violations. He also explained that many instances of private data leaking were non-malicious, so even as some application writers may attempt to circumvent the system, it may still be widely useful in the future.

StarTrack Next Generation: A Scalable Infrastructure for Track-Based Applications

Maya Haridasan, Iqbal Mohomed, Doug Terry, Chandramohan A. Thekkath, and Li Zhang, Microsoft Research Silicon Valley

Maya Haridasan presented her work on a service designed to manage paths based on GPS location coordinates. Mobile devices are now capable of recording their paths as a series of location and time tuples, which Haridasan calls a track. Once stored, these historical tracks can be a valuable source of information for users. Among many other examples, StarTrack could enable applications that provide personalized driving directions that take into account routes the driver is already familiar with. Using tracks from multiple users, ride-sharing applications could be developed.

To reach these goals, Haridasan and her team created the StarTrack service, designed to store and manage tracks and to provide a track-based programming interface. However, challenges arise in that tracks are error-prone, scalability is difficult, and applications require a flexible API. To address these concerns, Haridasan and her team developed a set of novel algorithms and data structures to compactly store, compare, join, and sort sets of tracks. After evaluating StarTrack against several other potential implementations, she closed by asking anyone interested in using the infrastructure to contact the authors.

Petros Maniatis from Intel Labs proposed a class of queries where information from multiple clients must be aggregated but not revealed. For example, the request "give me walking directions that don't intersect with my ex-partner" requires some knowledge of another's location, but ideally in a way that doesn't share private data. Haridasan and her team had considered similar use cases, but haven't found a solution yet. Michael Nowlan of Yale asked if there was any intention to associate intent or content with a track. Such a pairing could be used to separate driving tracks from other modes of transportation or to keep some paths private. Haridasan explained that there was already metadata associated with tracks, and ACLs could be used to preserve privacy, so such designs should already be possible. Stefan Sariou of Micro-

soft Research asked about handling errors and incorrect data. Haridasan replied that much of this was handled by the GPS location canonicalization algorithm, which converts a set of collected GPS samples into a path that goes through the underlying road network.

Virtualization

Summarized by Alan Dunn (adunn@cs.utexas.edu)

The Turtles Project: Design and Implementation of Nested Virtualization

Muli Ben-Yehuda, IBM Research—Haifa; Michael D. Day, IBM Linux Technology Center; Zvi Dubitzky, Michael Factor, Nadav Har'El, and Abel Gordon, IBM Research—Haifa; Anthony Liguori, IBM Linux Technology Center; Orit Wasserman and Ben-Ami Yassour, IBM Research—Haifa

Awarded Jay Lepreau Best Paper!

Muli Ben-Yehuda presented the Turtles Project, which was awarded one of two Jay Lepreau Best Paper awards for OSDI '10. The Turtles Project is an implementation of nested virtualization support for the Intel x86 architecture, which means it allows software written to use virtualization hardware support to run inside a hypervisor that already uses that hardware. Nested virtualization support allows for new applications, like hardware virtualization support for OSes that already are hypervisors (e.g., Windows 7 with XP mode) and deployment of virtual machines in the cloud. The x86 architecture supports only one level of virtualization in hardware natively, so multiplexing the virtualization hardware is necessary. The difficulty is to perform this efficiently, as changing the state used by the virtualization hardware requires expensive VM exits, and multiplicatively more exits per level of nesting.

Ben-Yehuda focused on MMU and I/O virtualization efficiency improvements. For MMU virtualization, he described three schemes in increasing order of efficiency. The key problem is that even with extra Extended Page Table (EPT) hardware support for more efficient translation, with any nesting depth greater than one there will be more translations necessary than hardware can provide, so translations must be compressed into fewer mappings. The most efficient "multi-dimensional" paging scheme compresses EPT mappings, since they change less frequently. For I/O virtualization efficiency, Ben-Yehuda described the most difficult case they had to tackle: the direct assignment of devices to nested VMs. Direct assignment requires an IOMMU for safety, but for nesting there is added difficulty, since the IOMMU itself must be emulated and requires analogous mapping compression.

The most important take-away measurements were approximately 10% CPU overhead per level of nesting, that multi-dimensional paging can be a several hundred percent performance win for page-fault heavy loads, and that multi-level device assignment can obtain equal device performance but at significant CPU cost. A significant portion of the added cost for multi-level device assignment could be removed if direct interrupt delivery to nested VMs was supported. The code was mature and efficient enough to be added to KVM.

An audience member from NICTA asked whether the performance would continue to get 10% worse per level of nesting. Ben-Yehuda responded that the exit multiplication effect would get worse as the level of nesting increased, and thus nesting performance would generally get worse by more than 10% per level with higher degrees of nesting. Nathan Taylor of the University of British Columbia asked about the security implications of this work, whether this would facilitate VM-based rootkits and whether I/O would remain safe with this implementation. Ben-Yehuda noted that OSes can easily detect when they are being virtualized (via timing irregularities, for example), so that VM-based rootkits are no less detectable with this work. He also said that trusted computing technology could potentially be employed to ensure the underlying hypervisor boots from a known safe environment. Sorav Bansal of IIT Delhi asked whether binary translation could help further reduce overhead, and Ben-Yehuda said this was possible but difficult for nested virtualization, due to lack of knowledge of hypervisor memory layout.

mClock: Handling Throughput Variability for Hypervisor IO Scheduling

Ajay Gulati, VMware Inc.; Arif Merchant, HP Labs; Peter J. Varman, Rice University

Ajay Gulati presented mClock, a new algorithm for scheduling I/O requests from VMs. He pointed out that controls on CPU allocation for VMs are fairly mature by now, but that there aren't equivalent controls for I/O requests (IOPS). In particular, reservation and limit controls of a fixed number of IOPS/time are not available, which is problematic in that storage is often from a shared network device, causing variable total capacity for all VMs on a host and making pure proportional sharing inappropriate.

Scheduling algorithms for VMs are often phrased in terms of time tags, in which the VM with the minimum time tag is scheduled. Gulati explained that the key points of mClock were real-time tags and separate time tags for reservations, limits, and proportional shares of resources beyond reservations. Real-time tags are used to ensure that the procedure can track actual rates of IOPS/time. Tags are prioritized so

that VMs not making their reservations take precedence and that VMs exceeding limits are not scheduled. The effectiveness of mClock in maintaining limits and reservations was demonstrated empirically with a graphical side-by-side throughput comparison. A separate enhancement allowing VMs to gain credit for being idle was made, which is important as I/O traffic is often bursty. mClock is general enough to be employed for other resources (such as network I/O) as well.

Etienne Le Sueur from NICTA pointed out that it appeared that the total IOPS/time dropped under mClock. Gulati said that the workloads of VMs in their experiments have different read-write ratios, degrees of randomness, and I/O sizes, so that the degree of variance Le Sueur observed (several hundred IOPS) was not unexpected. Gulati pointed to workload details on a backup slide that is reproduced in the paper. Another audience member asked how mClock balances latency and throughput tradeoffs. Gulati responded that mClock is more about dividing IOPS among VMs with reservation, limit, and share controls, but that prior VMWare work (PARDA) in FAST '09 dealt with latency control by throttling hosts. He said that for stronger guarantees, ideally one would have underlying hardware guarantees from vendors.

Virtualize Everything but Time

Timothy Broomhead, Laurence Cremean, Julien Ridoux, and Darryl Veitch, Center for Ultra-Broadband Information Networks (CUBIN), The University of Melbourne

Darryl Veitch presented an architecture for tracking time more accurately in the Xen VMM. The motivation for this work is that there are a number of applications—including finance, network monitoring, and distributed gaming—that require accurate timing information, but current timekeeping methods produce inaccurate results in VMs. Factors like clock drift combine with variable latency caused by other VMs in ways that can cause feedback mechanisms in ntpd, the current de facto standard for Linux timekeeping, to become unstable. Additionally, it is difficult to preserve correct times during live migration of VMs.

Veitch described his group's prior work with the Robust Absolute and Difference (RAD) clock, and he explained why it is a good fit for Xen. The key design point of RADclock is that it is "feedforward"-based. This means that system clock timestamps, which already have corrections applied, are not used to timestamp timing packets. Instead, raw counter values are used, and clock calibration is achieved through variables that convert raw counters into real times. VM clocks can then all read one hardware counter and calibration variables hosted in Dom0, which has hardware access; this is referred to as a dependent clock design. VM migration becomes easier in this design, since hardware counter values can be converted into new system clock times merely by using the calibration variables of Dom0 on the new machine rather than transplanting feedback-related state calibrated on one machine onto another whose counters have origins, drift, and temperature environment that are completely different.

The RADclock-based design was tested against Xen's current and prior timekeeping mechanisms. Veitch said that the measured error in standard RADclock operation appears to be low enough that the primary source appears to be air-conditioning flow in the room. Also, migration caused inaccuracy of the order of tens of microseconds in RADclock, as opposed to several seconds with current Xen timekeeping.

David Cock of NICTA asked whether a dependent clock design like RADclock could be used with feedback-based mechanisms without compensating clocks, as in Xen's current timekeeping. Veitch claimed that this would be difficult. Another audience member wanted clarification on what accuracies are achievable with RADclock. Veitch said that this depends primarily on characteristics of the connection to a time server. He pointed to his prior work showing that many kinds of load (e.g., high temperature) are not problematic, and estimated maximal accuracy in the range of tens of microseconds.