Conference Reports

In this issue:

Peter Bailey receiving the Computer Research Association Undergraduate Researcher of the Year award from Matt Welsh at HotOS XIII

13th Workshop on Hot Topics in Operating Systems (HotOS XIII)

Napa Valley, California
May 9–11, 2011
Sponsored by USENIX, the Advanced Computing Systems Association, in cooperation with the IEEE Technical Committee on Operating Systems (TCOS)

Opening Remarks

HotOS XIII Program Chair: Matt Welsh, Google

Summarized by Rik Farrow (rik@usenix.org)

Matt Welsh opened the workshop with a quick description of how it was structured. Each speaker had only 10 minutes for his or her presentation, with five minutes allotted for questions. Participants could interrupt the speaker during that 10 minutes. At the end of two sessions (five or six presentations), there would be a 45-minute discussion session, where the topics might involve the previous presentations, or anything else that was relevant.

At the end of the workshop, Matt announced some awards. (Yes, I know this is putting the cart before the horse, but I am not certain you will notice these announcements unless I put them up here.)

Matt Welsh, who came from Harvard to work for Google, announced that Peter Bailey, with whom he had worked for eight years at Harvard, had won a Computer Research Association Undergraduate Researcher of the Year award, which includes a 500-pound marble obelisk that had already been delivered to Peter, a certificate, and the support to attend the conference of his choice. One of Mike Freedman's students also won a CRA award this year.

Matt then told us who had won Google Chromebooks by their workshop presentations; Vijay Vasudevan (CMU) with his poster that took a position against the paper he presented; Dave Ackley (U New Mexico) for the most outrageous opinion, best expressed in person, but his paper does nearly as well. They decided to give two best talk awards, one to Mike

Walfish (U of Texas, Austin), who used Prezi, and one to Chris Rossbach (MSR), who got the remaining Chromebook, since Matt thought it would be easier to ship one to Austin than to Microsoft Research.

Putting the Hard Back in Hardware

Summarized by Derek Murray (Derek.Murray@cl.cam.ac.uk)

Mind the Gap: Reconnecting Architecture and OS Research

Jeffrey C. Mogul, HP Labs, Palo Alto, CA; Andrew Baumann, Microsoft Research, Redmond, WA; Timothy Roscoe, ETH Zurich; Livio Soares, University of Toronto

Jeff Mogul kicked off the workshop with a talk about a paper that arose from the "Research Vision" session at OSDI '10. The problem is that the computer architecture and OS research communities are drifting apart. New architectures are developed with little regard for the OS, which is considered to be so unknowable that it is a source of "noise" in benchmarks. This is largely due to the gold-standard benchmarks—such as SPLASH, SpecCPU, and PARSEC—which run almost completely in user mode for an extended period of time. By contrast, the state-of-the-art for measuring OS performance on an architecture is limited to little more than system call and page fault micro-benchmarks. A few semi-realistic benchmarks do exist, including SPECWeb and TPC-W, but they don't capture the full variety of applications that run in a realistic system.

The OS researchers in the audience weren't immune to Jeff's criticism. Our unquestioning dedication to developing systems that run on "commodity hardware" means that we are missing the opportunity to ask for new features. If we don't ask, we will end up with features that work when running a single HPC application but are incompatible with the isolation properties that an operating system must provide. For example, a platform might provide low-latency message-passing support using shared memory buffers, but sharing such a facility between multiple processes requires a kernel entry, which effectively erases the latency benefits. Leading by example, Jeff then presented his desiderata, which include cheap inter-core messages, lightweight inter-core notifications, faster syscalls, software-controlled caches, and better performance counters. A common theme was that the architecture shouldn't bake in policy (such as cache coherency) without providing the developer with an escape hatch to try different approaches.

Mike Swift (Wisconsin) opened the Q&A by asking how the OS community could improve its review process for papers that suggest architectural changes. Jeff replied that it is often difficult to assess whether or not a change is feasible, and the purpose of ASPLOS was to have a program committee with a range of experience to aid this. Gernot Heiser (UNSW/NICTA) remarked that some of the architectural critique—of IPIs, in particular—was x86-specific, since ARM and MIPS don't suffer from all of the same problems. John Ousterhout (Stanford) asked how we can make incentives for architecture people to make changes, and pointed out that we should be careful to distinguish between the research community and the people who actually build the hardware. Jeff pointed out that one of the hurdles is that new architectures usually need to run a commodity OS, so there is a chicken-and-egg problem. Finally, Erez Zadok (Stony Brook) lamented that many architectures have a wide range of performance counters in hardware, but OEMs selectively disable many of them in the BIOS. Jeff remarked that some OEMs might be receptive to changing this.

Operating System Implications of Fast, Cheap, Non-Volatile Memory

Katelin Bailey, Luis Ceze, Steven D. Gribble, and Henry M. Levy, University of Washington

Katelin Bailey said that the real-soon-now advent of fast, cheap non-volatile RAM (NVRAM) may have a disruptive effect on OS design. Many of the assumptions in current OS design are based on a two-level memory hierarchy of fast DRAM and slow disks; NVRAM threatens to shake things up, because it potentially combines the speed of DRAM with the persistence of disk—i.e., it offers the best of both worlds. Existing research has focused on incremental steps, such as replacing disk with NVRAM and retaining file system semantics, or using virtual memory to build single-level store that combines RAM and DRAM. But this isn't radical enough: how about replacing *all* of the memory in a system with NVRAM?

Such a system would have many desirable properties. For example, hibernation and reboot would become extremely efficient, because there would be no need to copy state to or from a secondary storage medium. The very fast write performance would also make deterministic record/replay techniques much more practical. However, there are a number of challenges that would need to be addressed: for example, if your entire system image is persistent across reboots, how would you deal with bugs and rolling back to a known-good state? How should sensitive data be treated, now that it could persist for a much longer time? Furthermore, current virtual memory techniques were originally developed with the dual role of enabling swapping (which is no longer necessary) and protection (which is), so the development of a system with

NVRAM everywhere would provide a good opportunity to rethink the assumptions about granularity, for example.

Katelin admitted that the talk raised more questions than it answered, but the audience was on hand to raise even more. John Ousterhout (Stanford) harked back to the 1970s, when every computer effectively had NVRAM in the form of core memory, and he pointed out that nothing changed when DRAM displaced core. Katelin replied that it's still worth exploring our options. Mike Swift (Wisconsin) and Joe Tucek (HP Labs) raised the smartphone question, asking what we could learn from those platforms, but Katelin said that the approaches taken on those devices are relatively conventional. At this point, a waggish audience member pointed out that cell phones reboot every time daylight savings time happens, so they're not there yet. Mothy Roscoe (ETH Zurich) went Back to the Future, pointing out that many of these ideas had been tried before in systems like KeyKOS and Multics, but they hadn't caught on. Katelin said he hoped that fast NVRAM should enable us to do things that weren't possible in those days.

Virtually Cool Ternary Content Addressable Memory

Suparna Bhattacharya, IBM Linux Technology Center and Indian Institute of Science; K. Gopinath, Indian Institute of Science

Suparna Bhattacharya rounded off the hardware session by discussing another exotic form of memory: ternary content-addressable memories (TCAMs). Their associative addressing means that TCAMs have seen a lot of use in caches and high-performance routers, but more exotic uses have been discovered, such as encoding deterministic finite automata, ternary Bloom filters for subset matching, and similarity search algorithms. With progress at this rate, we can expect the range of applications to grow to the point where application developers may want to harness TCAMs, so this talk looked at ways that virtual memory techniques could be used to provide the illusion of vast amounts of associatively addressed memory.

It isn't feasible to build a single huge TCAM, because power consumption and latency increase with the number of keys. Therefore, Suparna discussed various ways that a virtual TCAM could be built from a combination of TCAM and DRAM. The basic idea is to build a cache hierarchy, with the level-1 store implemented in a TCAM and the level-2 store simulating associative lookup in DRAM. The first challenge is choosing a replacement strategy (and an application) that exploits temporal locality, so that as many lookups as possible are served from the TCAM. Spatial locality is less important (since in an associative store, location is not important), but there are some potential wins to be had by compressing keys using the don't care bits. This approach is particularly beneficial if there is "content locality," i.e., keys in a similar range frequently accessed together. Finally, it will be challenging to build an efficient TCAM simulator that uses DRAM, since the don't care bits mean that standard search algorithms do not apply; one possibility is to use part of the TCAM to store a mapping from partitions of the key space to DRAM locations. Suparna ended by echoing the previous talks in this session: it would be useful to engage the architecture community in order to develop TCAMs that are better suited to general programming—for example, by providing better support for multiple matches. She also speculated that the availability of NVRAM might open up new possibilities for TCAMs.

Jeff Mogul (HP Labs) asked how Internet routers—which must store the entire routing table—deal with a limited amount of TCAM, and whether they use similar techniques. Suparna replied that most routing tables try to do static compaction using the don't care bits, but she agreed that there may be tricks that could be picked up from these devices. Mike Freedman (Princeton) asked about applications and whether in this model TCAMs would be part of the general-purpose memory hierarchy. Suparna replied that the intention was to expose (virtual) TCAMs to applications as general-purpose memory, and that there were many search-based applications—for example, in data mining—that could benefit. Matt Welsh (Google) brought the session to a close by remarking that GPUs had become commonplace thanks to 3D gaming, and TCAMs might have a similar "back-door" application that pushes them into widespread use.

Soft Fluffy Clouds

Summarized by Derek Murray (Derek.Murray@cl.cam.ac.uk)

The Best of Both Worlds with On-Demand Virtualization

Thawan Kooburat and Michael Swift, University of Wisconsin—Madison

Thawan Kooburat enjoys the advantages of virtualization, but he's concerned that its constant overhead is inhibiting adoption, especially in large datacenters at Google and Facebook, and on resource-constrained devices like your laptop. The idea of "on-demand virtualization" is that you only pay the cost of virtualization—both in terms of performance overhead and limited functionality—when its features are going to be used. Therefore, most of the time the operating system uses native execution, then it slips into virtualized mode *on demand* when the user wants to migrate execution, checkpoint the system state, and so on.

Thawan described how on-demand virtualization is implemented. The basic technique is to use the OS hibernate function (implemented using the TuxOnIce patch to Linux 2.6.35) to create an image of the system state, and then transfer that

state into a virtual machine (implemented using KVM). One challenge is that the native and the virtualized hardware profiles will likely be different, with the VMM typically providing a feature set that lags behind native functionality. This is addressed with device hotplug and another level of indirection: logical devices that retain all necessary state and hide the hotplug events from the applications that use these devices. Thawan has a prototype that currently supports one-way conversion from physical to virtual, which takes approximately 90 seconds and succeeds without closing an open SSH connection. Future improvements will include hibernate-to-RAM, which will improve performance, and performing the virtual to physical conversion.

Mike Schroeder (Microsoft) asked if this defeated the purpose of virtualization as a means of providing a defense against security issues. Thawan replied that this is not the aim of on-demand virtualization, which is geared more towards migration and checkpointing. Peter Honeyman (Michigan) asked where to expect the crossover point when the cost of re- and devirtualization becomes greater than the cost of running permanently on a VMM. Thawan answered that it would be workload dependent. Philip Levis (Stanford) raised a concern about what would happen when migrating an OS that used a large amount of local storage on native disks, and Mothy Roscoe (ETH Zurich) pointed out that a paper at the last HotOS had solved the apparently harder problem of migrating between two physical machines with no virtualization involved.

Repair from a Chair: Computer Repair as an Untrusted Cloud Service

Lon Ingram, Ivaylo Popov, Srinath Setty, and Michael Walfish, The University of Texas at Austin

Michael Walfish is dissatisfied with the status quo in computer repair. Today, it resembles television repair, whereby you bring your computer to a retail service that is both inconvenient and insecure. Solutions based on providing remote desktop access are not ideal, because you have to monitor every action by the repairer, or it will be just as insecure as taking your computer to a shop. In this talk, Michael presented "repair from a chair," which uses virtualization technology to make the software components of a computer available to a repairer in a secure fashion. An in-depth study of Geek Squads, Genius Bars, and IT services at UT Austin revealed that the vast majority of repairs are software-only, and so this would be a feasible solution.

The system includes a module called the "repair helper," which lives between the OS and the hypervisor to facilitate repair. According to the paper, the main function of the repair helper is to migrate a copy of the VM to the repairer with private data scrubbed; there is a large design space to explore here. In the talk, Michael focused on the idea of using "action graphs" to represent the changes made by a repairer, and hence provide integrity guarantees. The hope is that repairs could be encoded in a canonical representation, which could then be signed by the repairer for assurance and auditing purposes. The action graph representation would also help to maintain availability of the machine while under repair: the customer could continue to use the machine, and changes by the customer and the repairer could be merged using a process that is analogous to git rebasing.

The talk provoked a lot of discussion and was awarded one of the Best Talk prizes at the end of the workshop. Mike Swift (Wisconsin) was first up to ask whether on-demand virtualization (from the previous talk) would be ideal for this. He also had a real question about what fraction of repairs would be difficult to handle, and how hypervisor device driver problems might be handled in the cloud. Michael replied that configuration errors would be in scope, but he hadn't considered hypervisor issues, since it was assumed that the customer wouldn't (or wouldn't be able to) mess with the hypervisor configuration. Jeff Mogul (HP Labs) took a different tack, suggesting that, if all the repairs were canonical and could be signed, the repair service could just apply all known repairs indiscriminately. Michael countered that there might still be some human intelligence required to choose the correct ordering. Then Jeff raised the specter of having to trust "canonical compositions," but Michael replied that this is not necessary if there is an auditable log. Finally, Brad Chen (Google) characterized this as an "automatic update" problem, and asked whether this would cease to be a problem when applications are cloud-based. Michael replied that as soon as devices become used for content creation, rather than consumption, configuration issues will start to arise again.

This marked the end of the formal Q&A, but this talk was the subject of much debate in the discussion/open mike session that follows below.

Structuring the Unstructured Middle with Chunk Computing

Justin Mazzola Paluska, Hubert Pham, and Steve Ward, MIT Computer Science and Artificial Intelligence Laboratory

Justin Mazzola Paluska gave an intriguing talk about a new construct that promises to unify parallel programming for GPGPUs, massively multicore systems, clusters, and clouds. At present, the structures used to represent programs and the structures of different execution platforms are orthogonal, and unstructured assembly code does a poor job of åtaking advantage of different, very specialized machines. "Chunks" are the solution: a chunk is a fixed-size block in memory

that abstracts program structure, and chunks are mapped individually onto machine structure. Each chunk has a fixed number of slots, each of which is fixed size. Each slot is typed, and it can contain a scalar value or a link to another chunk. One idea is that making links explicit exposes structure in the chunk graph, and the developer is forced into this by the relatively small size of a chunk.

The chunk graph creates many opportunities and challenges for improving parallel programs. First, a link is allowed to cross architectural boundaries, and chunks can migrate between processing elements, which helps in a heterogeneous multicore system. However, this creates a distributed garbage collection problem and requires a policy to decide which chunks should be migrated. Another feature of the model is that threads start out being represented by a single chunk with a link to a (possibly linked-list) stack of chunks, which in turn may be linked to function chunks or object chunks. The links can be used to compute a distance and size metric within a given thread, which helps the system decide which chunks should be co-located. For example, a distance-k neighborhood of the thread object would indicate the important chunks to co-locate, and overlapping neighborhoods would enable synchronization and contention to be inferred. The main hope, however, is that there will be many distant threads that can run without interference and can be scheduled to avoid false sharing and contention.

Dave Ackley (New Mexico)—who would go on to make a name for himself at the workshop with an outrageous programming model of his own—was concerned about the chunk graph turning into a "huge ball of high-dimensional goo." Justin countered that unused chunks would not need to be loaded in, and NVRAM could be useful to help with this. Aleks Budzynowski (UNSW/NICTA) was more worried about the amount of policy that seemed to be going on at the OS level, and would prefer to see more work being done at the language level or in the compiler. Justin replied that this is another way to experiment with the same issues, and the chunk model is an attempt to force the compiler into giving the OS something to which it can usefully apply policies. Dave Holland (Harvard) saw this as a graph clustering, which is a known hard problem, but Justin replied that hopefully he doesn't have to solve the general problem, if it is possible to use some heuristics at runtime, such as sending paths around. Finally, Mothy Roscoe (ETH Zurich) was unconvinced that there is a one-size-fits-all solution for the huge number of different scales, but Justin said he'd had positive experience with cloud and cluster computing (which is the easiest experimental platform). The reason for a one-size-fits-all solution is that Justin had seen schematic pictures resembling chunk graphs over and over in different venues, and he wanted to extract some common abstraction that could be useful.

Discussion/Open Mike

Summarized by Derek Murray (Derek.Murray@cl.cam.ac.uk)

By now the audience was fired up, and Matt Welsh (Google) opened the floor to anyone with something to say. Matt used chair's prerogative to make the first point about NVRAM: he likes the ability to wipe a computer's memory on reboot, because it's the only way to get it to a known-good state. Katelin Bailey (Washington) replied that rebooting wouldn't go away in the non-volatile future, but the aim was to separate the notion of resetting from the power cycle. Jeff Mogul (HP Labs) pointed out that this is a perfect example of decoupling mechanisms that don't belong together, as he had proposed in the first talk. Dave Andersen (CMU) was worried about the effect of random bit flips, but Margo Seltzer (Harvard) said that these are very unlikely in practice.

Geoff Challen (SUNY Buffalo) remarked that it was good to see many hardware people in the audience, which should help to address Jeff Mogul's criticism that the communities don't talk anymore. Mark Hempstead (Drexel), a self-confessed computer architect, announced that it was great to see a move towards better communication between the communities, and he asked people to send him C code that he could run. Mark raised a bone of contention: his aim is to have as few cycles in the OS as possible. Mothy Roscoe (ETH Zurich) disagreed, saying that many applications intentionally spend a long time in the OS, and this illustrates what hardware designers don't understand about operating systems. Mothy's real desire is hardware that does less stuff in hardware and just provides fast mechanisms that software can use. Jeff Mogul agreed with Mothy, telling Mark that, for example, fast cache coherence in the hardware is all very well, but sometimes there is a better policy for a given workload, and it would be desirable if we could implement that in software without having to trick the hardware into doing our bidding. Steve Hand (Cambridge) reminisced about the glory days of software/hardware co-design and mused that Intel should buy Microsoft or vice versa, to take us back to those days. Steve also praised FPGAs, which have become relatively easy to program, thereby allowing more people to try their hand at hardware design. Joe Tucek (HP Labs) mourned the loss of software-controlled TLBs. Margo Seltzer suggested that we need to pitch to industry, rather than other researchers, and asked what the virtualization researchers did to get hardware support in modern instruction sets. Matt Welsh—tongue firmly in cheek—suggested that the answer was to build something that is useful but really slow without hardware support.

Aleks Budzynowski (UNSW/NICTA) turned the discussion to repair-from-the-chair, asking whether anything had been

done to cut down the amount of state that must be sent to the repairer. Michael Walfish (UT-Austin) replied that techniques based on selectively faulting-in state to the repairer would work. Mike Swift (Wisconsin) was more attached to the idea of remote desktop solutions, but Michael replied that protecting against a malicious repairer was the real aim of the project, and where that was implemented really didn't matter. Mike Freedman (Princeton) suggested that the Geek Squad could provide a piece of software for the customer to install, which could be configured to allow access to different settings, but Michael was concerned about the cognitive overhead of configuration on non-technical users. Dave Andersen reckoned that the problem could be solved by Microsoft engineering a better access policy control panel. Mothy saw it more as a problem of liability if somebody were to make a mistake, and the "right answer" would only be found by talking to financial and legal people.

Petros Maniatis (Intel) took Mothy's point about non-technical issues and brought us back to discussing architecture. One of the overriding concerns for a processor company is whether adding a feature will get the company sued or cause bad PR. Steve Hand also mentioned the issue of backwards compatibility, which is often necessary and can inhibit innovation. Brad Chen (Google) suggested that our job is to discover technical choices, and present them to the business people; he mentioned Android and ChromeOS as two very different solutions to similar problems. Dave Holland (Harvard) pointed out that lawyers are trained to look for risk and not make policy, so we shouldn't worry about that so much, although Matt Welsh replied that that is easier to say in a university. Mike Freedman ended the discussion by remarking that he had spoken to a number of law professors who are in favor of technical solutions, because things are much slower to change in the legal and policy fields.

Panel: Cloud Computing

Panelists: Mendel Rosenblum, Stanford; Rebecca Isaacs, Microsoft Research; John Wilkes, Google; Ion Stoica, UC Berkeley.

Summarized by Lon Ingram (lawnsea@cs.utexas.edu)

John introduced the panel session by saying that Matt Welsh had asked them to fight, but they found that they agreed too much on the fundamental academic questions in cloud computing to do so. The panel chose to instead discuss two subjects that they did disagree on: (1) what should academics do that is not useless and (2) what should industry do that is not worthless.

The panelists offered brief introductory remarks, and John completed the introductions with a bid for the Most Controversial Opinion prize by declaring that he never wanted to read another paper submission that talks about improving Hadoop performance by 10%. Discussion then began in earnest, with the panel taking questions from the audience.

The conversation covered a broad range of topics, but a recurring theme was multiple pleas from academics for industry to release large anonymized datasets that would be useful for understanding what workloads datacenters see at scale. The industry representatives responded that this was unlikely to happen and that anonymizing such datasets is far harder than one would expect. Rebecca proposed a possible solution: academics should run their own commercial cloud platform as a way to generate such datasets themselves.

The panel and the audience also discussed the difficulties academics face evaluating proposed solutions without access to the kind of scale that industry sees. Mike Freedman of Princeton asked for examples of algorithms that looked good in the small but failed at scale. John replied that it typically isn't O(n) that is the problem but, rather, the complications introduced by interactions with other components and operational concerns—upgrading a system while it's in operation, for example. Ion added that academics need to understand how to evaluate solutions without running them at scale.

Matt Welsh from Google launched the final discussion of the session by asking how to get industry to open up more and how industry can help train the next generation. John suggested that those working in industry should find an academic and tell them about a problem they have—talking to them until they understand the problem.

We're Going to Need More Wine

Summarized by Srinath Setty (Srinath@cs.utexas.edu)

Macho: Programming with Man Pages

Anthony Cozzie, Murph Finnicum, and Samuel T. King, University of Illinois

Anthony Cozzie started the talk by pointing out a hard truth about programming: programming is hard and programmers make errors when they write code. Then he described the architecture of Macho, a system that can automatically generate Java programs. Macho takes the description of the functionality in a natural language as input and then uses a database of code snippets to stitch together a piece of code with the functionality specified in the natural language. Macho also includes an automated debugger to test the generated code using a set of examples.

Margo Seltzer from Harvard asked about the progress made in the project. Cozzie acknowledged that the problem is hard

and the module involving the database is the hard problem. Brad Chen from Google suggested that it would be very useful if Macho generated a specification along with the implementation. Cozzie agreed. Joe Tucek (HP Labs) asked about the amount of time taken for generating code. Cozzie replied that the ls example takes about 20 minutes.

Pursue Robust Indefinite Scalability

David H. Ackley and Daniel C. Cannon, The University of New Mexico

In the second of the two Best Talks, David Ackley pointed out the conflict between efficiency and robustness in computer systems. He went on to propose a computational model, Movable Feast Machine, to achieve indefinite scalability. However, this approach sacrifices the following three properties in the current system's architectures: first, fixed-width addresses and unique node names; second, logarithmic global communication cost; and third, clock and phase synchronization.

In the proposed design, the Movable Feast Machine consists of a 2D grid in which each tile contains a processor with a fixed amount of volatile and non-volatile memory. Each processor can communicate with its nearest neighbor processors via point-to-point links. The computation model for the proposed machine consists of a set of "event windows" that involve a group of tiles communicating with each other to perform the computation. Note that many non-overlapping event windows can exist concurrently. One of the critiques for this proposed architecture is that the hardware costs will be too high for cost-effective computation.

Mike Dahlin (University of Texas at Austin) asked about the rationale behind choosing small atom sizes. Ackley answered that the smaller sizes provide fine-gained mobility, which is essential for indefinite scalability. Dave Anderson (Carnegie Mellon University) asked about the advantages of Movable Feast Machine's local propagation restriction. David replied that local propagation enables expressiveness in the proposed architecture. Erez Zadok (Stony Brook) asked whether he had looked at any newer computing models to see whether anything matched. David replied that his PhD work was in neural networks a thousand years ago, and this is his attempt to start again from scratch. Michael Walfish (UT-Austin) asked about the types of computations that can be represented in the proposed computation model. Dave said any computation under the stochastic flow-sorting category can be represented on Movable Feast Machines. Toby Murray (NICTA/UNSW) asked if there is a way to quantify the error in output generated for the computations run on the proposed architecture. David acknowledged that quantifying error is a hard problem and one could reduce error by replication and repetition.

Hear Ye, Hear Ye

Summarized by Suparna Bhattacharya (suparna@csa.iisc.ernet.in)

*Benchmarking File System Benchmarking: It *IS* Rocket Science*

Vasily Tarasov, Saumitra Bhanage, and Erez Zadok, Stony Brook University; Margo Seltzer, Harvard University

Vasily Tarasov began his talk by citing a recent study which found that research conclusions in medicine often contain misleading findings with a heavy focus on exciting results to the exclusion of other aspects, and noted that similar observations could be made about the state of filesystem benchmarking. He argued for an improved evaluation approach which adequately reflects the complex multi-dimensional character of file-system behavior. As a follow-up to their previous ACM TOS (Transactions on Storage) paper, "A Nine-Year Study of File System and Storage Benchmarking," he told how he and his co-authors surveyed 100 file system papers from 2009 and 2010 and found a wide range of benchmarks used, with little standardization, e.g., as many as 74 ad hoc benchmarks and 24 custom traces. Even among the standard benchmarks used, many were based on compilation or small file operations, effectively stressing CPU or memory more than on-disk layout.

As a possible way forward, Vasily proposed creating standardized benchmarks for common filesystem dimensions such as on-disk layout, prefetching, and in-cache performance. In addition he emphasized the need for reporting results in terms of curves and distributions across a range of parameters instead of a single number, since filesystem behavior can be sensitive to even small changes in the environment. To illustrate how widely conclusions from benchmarking may be impacted by the choice of evaluation approach, he presented an interesting case study comparing the graphs of random read throughput (using filebench) of a 410 MB file across three file systems (ext2, ext3, and XFS) as measured at 10 second intervals. Initially, the performance is I/O-bound and eventually, when the file is completely in the page cache, it becomes CPU/memory-bound. At both these extremes, performance is similar for all three file systems, but in the transition range, which involves a 10-fold jump in throughput between the interval from 200 to 800 seconds, the differences between file systems can vary widely (up to as much as an order of magnitude) depending on the time when measurements are made. This can result in radically

different conclusions from point comparisons. Likewise, a 3D plot of latency histograms collected periodically for ext2 random reads reveals a bimodal kind of characteristic, with a 1000-fold difference between the modes. Average results make very little sense in such situations.

Someone raised the concern that it might be very tough to ensure that the dimensions are orthogonal to each other. Vasily responded that indeed isolating dimensions is important but sometimes hard; however, even without orthogonality, we can still ensure coverage. Phil Levis (Stanford) felt that the comparison with medicine might be misleading since, unlike medicine, file systems do not involve human subjects.

John Ousterhout observed that the real question is not just one of capturing data but a need to understand and explain what is actually going on, e.g., the reason for different modalities in the graph. In the ensuing discussion, Jeff Mogul argued that the purpose of benchmarking is comparison, not understanding. With this approach it isn't clear how one would compare these multi-dimensional result distributions. Margo Seltzer responded that there is no one unidimensional comparison that works, because the weighting may not be same for all uses. While this means more work for the reader, it ensures that results are less biased. Erez Zadok observed that the networking community uses CDFs more than the storage community. Jeff Mogul asked whether they explain why one CDF is better than another. Margo Seltzer reiterated that there is no single preferred answer; it depends on what we are trying to achieve. Vasily observed that often such benchmarking really comes down to benchmarketing. Someone remarked that having a marketing target can help, especially in pushing improvements over time, and asked whether we should redefine benchmarks as a composition of performance curves and a purpose-specific utility function.

David Holland remarked that as a consumer of benchmarks we still don't know what the good choices are. The state of file-system (FS) benchmarking in the OS community is abysmal; we need an official set of FS benchmarks. Erez Zadok responded that among benchmarking tools, they found filebench to be nice and hence forked and fixed it. Now it supports two dozen random distributions, can handle multimodal distributions, and uses a data generator instead of merely writing zeros as some other benchmarks do.

Multicore OS Benchmarks: We Can Do Better

Ihor Kuz, ETH Zurich, NICTA, and the University of New South Wales; Zachary Anderson, Pravin Shinde, and Timothy Roscoe, ETH Zurich

Ihor Kuz observed that there is a fundamental problem with existing multicore OS benchmarks—they measure scalability of applications but do not evaluate how well the OS manages performance isolation between different applications. In this respect, they fail to expose performance effects of what is arguably the central purpose of an OS: that of allocating and sharing resources across applications. To address this concern, he proposed a systematic benchmarking approach that employs a mix of application workloads running concurrently. The mix is carefully chosen in a way that (1) exercises multiple system resources without overcommitting any resource and (2) is performance-sensitive to the availability of resources.

An application-specific goodness function is used to perform a sensitivity analysis of the performance of each candidate application variant (choice of application parameters) with respect to various machine resources: e.g., CPU, cache, memory, disk, and network. For example, a Web browser is partly sensitive to network bandwidth, with the rendering of Web pages being CPU-sensitive. On the other hand, the goodness metric of a virus scanner might be the number of files scanned, which is disk-bound. The design of the optimal (maximally sensitive) mix is posed as an integer linear programming optimization problem, based on resource usage and sensitivity, subject to the constraint of avoiding resource overcommit. Intuitively, the optimal solution is a mix of application variants that use resources that they are most sensitive to. Once the results from running an optimal mix on an OS have been obtained, several evaluations may be performed. For example, the performance difference in running an application unmixed and mixed can highlight potential problems in the system. Different operating systems might have a different optimal mix; comparing performance at these points can indicate how well each OS manages its optimal mix.

Ihor concluded with some comments on the status of the work. Currently they have tried this with Linux microbenchmarks; they need to run it with real applications. The approach assumes a constant resource usage, hence will need to be extended to account for bursty applications. Further, it uses a static mix, while in desktop scenarios, application mixes are dynamic.

Michael Dahlin asked whether the optimal mix gains in stressing the OS while sacrificing completeness, making it difficult to compare results (unlike typical OLTP benchmarks). What if the optimal mix is not even realistic? Ihor responded that one can play around with parameters and constraints of the ILP formulation to restrict solutions to realistic or sensible combinations rather than irrelevant mixes. Livio Soares suggested including OS abstractions of resources in addition to raw resources and Ihor agreed. Someone raised a concern about the difficulty of stating and proving that various resources can be scheduled together

without overcommit in tricky situations. Ihor accepted that one may run into this issue for real applications but observed that OSes need to handle such situations, so we should be able to test for this.

It's Time for Low Latency

Stephen M. Rumble, Diego Ongaro, Ryan Stutsman, Mendel Rosenblum, and John K. Ousterhout, Stanford University

Steve Rumble made a case for rearchitecting systems for low latency communication in datacenters, anticipating realizability of up to two orders-of-magnitude improvement in RPC round-trip times and the significant impact this can have on enabling future Web applications. He began by highlighting the increasing demand for low latency as foreseen by constraints faced today by applications like Facebook, which randomly access many pieces of non-local interdependent data in fast DRAM-based storage for each small request. Commodity network bandwidth has increased by a factor of 3000 in the past 30 years, while latency has only decreased by a factor of 30; high latency limits Facebook to 100–150 dependent data accesses per page request. Working around such constraints not only adds to application complexity but also renders certain features non-viable.

Steve then presented a component-wise breakup of the high 300–500us RPC latency in current datacenters. He observed that because of the small distances between servers within a datacenter, the limiting factor is not the propagation delay (< 2us) but the delays across multiple hop switches (10 hops with 10–30us/hop) and a comparable delay in the NIC (10–128us) and OS stack (60us). He then argued that recent hardware improvements have brought us to the cusp of low latency. The time is right for the OS community to initiate a rethinking of the stack and architecture to reduce the rest of the overhead. 100ns latency switches and 1us latency NICs are already available in the HPC space, with Fulcrum Microsystems and Mellanox pushing the boundary to sub-500ns switches in the commodity Ethernet space. Steve predicted that this means that 5–10us round-trip times are within reach in the short term by addressing OS/protocol overheads while defining a simpler API structure that has a different distribution of responsibility between the OS, application, and the NIC than Infiniband/RDMA or U-Net. Since a datacenter is a closed ecosystem, it is even possible to experiment with new protocols that can scale low latency to 100K+ nodes instead of living with TCP. Steve projected that even lower latencies are possible in the long term; below 10us, transferring data between the NIC and the system would become a bottleneck, but a round-trip latency of 1us is achievable in 5–10 years by re-architecting systems to transmit/receive data directly from the CPU cache.

The talk generated a lot of questions both during the Q&A and the discussion session that followed. Matt Welsh remarked that we've been through similar work in the past which failed, but not due to technical reasons—the NIC was the bottleneck back then too. Could those ideas (from active messaging/U-Net) be applied now or is there something fundamentally different? The response was that today we have massive datacenter applications that need this, and low latency is becoming practical in commodity space. Matt followed up by noting that we knew how to get good performance under ideal conditions but the programming model at that time was awful—the sheer amount of engineering needed to get stable performance over time was a challenge. It was mentioned that many SIGCOMM papers had appeared on the chained RPC and scatter-gather problem, but no one cared before. Further, while we can make it faster, maybe XML/SOAP is not the most efficient way to dispatch requests—we will run up against the propagation wall sometime. Mike Schroeder noted that commodity support does not matter all that much, since there is a need within a datacenter. He mentioned that they were seeing problems with packet switching and might need circuit switching instead. John Ousterhout remarked that the community was too influenced by the success of MapReduce, which is bandwidth -oriented; there are other applications, such as realtime analysis of graphs with no locality, that really need low latency.

There was a question about why the DRAM isn't directly put on the NIC, since the CPU is not really used; Steve responded that it is essentially the same, only the CPU is programmable. Joseph Tucek remarked that infiniband costs only $300/port which is not that expensive. Michael Swift wondered if there was a case for saving data persistently at low latency, especially with NVRAM, but no suggestions came up. Michael Dahlin observed that the fact that there are about 150 dependent data-access steps for a Facebook request was intriguing, and asked whether it was the ratio of latency to overhead that mattered and if benchmarks could be designed to capture this. Perhaps the cool stuff did not matter because it got hidden by other overheads. Prabal Dutta asked why the netFPGA project was not considered a fabric to explore these questions. Steve responded that he didn't think that switches are an issue and that future problems in NICs will only appear after we solve the other problems to get to 10us latency. Timothy Roscoe commented that the problem is not the design of NICs (modern NICs are pretty good), but in interfacing with the application after it gets the data, especially as NIC latencies are getting close to DRAM latency.

Discussion/Open Mike

No report is available for this session.

Data Still Matters

Summarized by Vasily Tarasov (tarasov@vasily.name)

Disk-Locality in Datacenter Computing Considered Irrelevant

Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica, University of California, Berkeley

Ganesh Ananthanarayanan talked about the changes in the notion of disk locality in data-intensive computing. Disk locality is exploited at all levels of the storage stack: applications, file systems, disks. The fundamental reason why corresponding optimization methods work is that disk bandwidth is significantly larger than network bandwidth. However, this statement is less true nowadays. Off-rack, rack-local, and local disks all perform almost the same. Designers still need to care about RAM locality, however. But datasets are huge in data-intensive applications (e.g., 200 times larger than available RAM size in Facebook). Can anything be done about that? It turns out that for 96% of the jobs, all the required data can fit in the RAM. It is just that current caching policies (such us LRU) cannot predict well which data to put in the RAM. Ganesh concluded that software needs to be more intelligent in deciding which data to put to the cache and which to evict.

Gregory Ganger of CMU said that this is a great example of a collective action problem. If everybody ignored caching, this would place tremendous demand on the network. Ganesh replied that in any case the network bandwidth is so high that disk locality becomes less important. Someone asked what were the 96% of jobs doing? Maybe they were CPU-bound, and Facebook just runs applications incorrectly? Ganesh agreed that it would be great to have this information but they do not have it. Someone commented that it is very easy to get a one-rack node with 12 locally attached disks. The author responded that according to his calculations one needs at least 50 disks per node.

Optimizing Data Partitioning for Data-Parallel Computing

Qifa Ke, Vijayan Prabhakaran, Yinglian Xie, and Yuan Yu, Microsoft Research Silicon Valley; Jingyue Wu and Junfeng Yang, Columbia University

Qifa Ke discussed intelligent data partitioning for performing distributed computations. When one needs to run some computation across several nodes, the job needs to be divided between these nodes. As part of this process, the data needs to be partitioned. What is the optimal number of partitions and the partition function? The simplest (and quite common) method is to partition data using a hash function, but data skew can happen in this case. In addition, computation skew can occur; even if data is of the same size, computation time is not the same for all partitions. Moreover, balanced workload does not always mean optimal performance, and the authors worked on determining the best partitioning scheme given the data and application. Data is not structured in this case, unlike with databases, which makes the problem more difficult. The authors were looking for a compact data representation. Code is user-defined as well, with different languages and execution modes. The authors proposed a three-stage approach: model the partition scheme, estimate performance, and find the scheme that provides optimal performance.

Darren Martin of Cambridge asked whether the authors plan to do partitioning online or offline. Qifa replied, Both. Another person asked what happens if the optimal solution is 50 partitions but one has only 49 nodes. Qifa said that at the moment they consider only an ideal case. Somebody suggested they use AI techniques for the partitioning problem.

Disks Are Like Snowflakes: No Two Are Alike

Elie Krevat, Carnegie Mellon University; Joseph Tucek, HP Labs; Gregory R. Ganger, Carnegie Mellon University

A lot of today's systems and techniques rely on the idea that two identically labeled pieces of hardware will perform identically, but this is not true anymore. Elie Krevat presented a study showing that modern disk drives, even if their makes and models match, perform differently. In this study, the authors looked at three generations of disk drives: 2002, 2006, and 2008 vintages. The throughput of disk drives produced in 2002 was the same. In 2006 the variance in performance reached 10%, and it increased to 20% in 2008.

The reason for this behavior is a new and "almost undocumented" feature of disk drives: adaptive zoning. Zone Bit Recording (ZBR) has been around for many years. This technology allows vendors to put more sectors on the outer tracks of a platter. However, before now, zones' boundaries were fixed by the specification of the disk. Now, on the other hand, disk manufacturers test every individual read-write head with respect to the data rate that it can sustain. Then they assign zone boundaries according to this information. Interestingly, this happens even within the disk: different platters have different zoning.

Margo Seltzer said that her research group has been aware of similar problems with other components for a long time. But nobody cared. Why should they care now? Elie responded that there are systems that can neglect this, but others should care. Michael Schroeder of MSR pointed out that pundits say that the rise of SSD drives will make this a moot point. Elie

responded that SSDs are still quite expensive and that even flash drives can have variations in their performance. Philip Levis questioned if the cache will amortize this problem. Elie agreed that this can happen as long as you can prefetch data in time. But in the paper, the authors used different, streaming workloads.

Watts Up, Joules?

Summarized by Vasily Tarasov (tarasov@vasily.name)

The Case for Power-Agile Computing

Geoffrey Challen, MIT, SUNY Buffalo; Mark Hempstead, Drexel University

This presentation was unlike any other talk at the workshop—it was a whole show! I'll try to describe it, but you really had to be there to truly appreciate it. First, the title slide appeared but the presenter seemed to be missing. After a period of growing uncertainty in the audience, the second author, Mark Hempstead, stood up and said that he would have to give the talk, but that he had not seen the slides before. He pressed the space button on the laptop and what the audience saw on the screen was a genie lamp. Over the laughs of the crowd, Mark humbly confessed that he was not aware of the purpose of this slide. Maybe we need to rub the lamp in order for a genie to appear, he said. He tried it... and Geoffrey Challen, first author, ran into the room in a golden hat and a vest over his naked torso shouting "Shazam! Shazam! Shazam!" The audience roared.

The rest of the talk was a conversation between the genie (Geoffrey) and the genius (Mark) during which they designed extremely power-efficient systems that can scale to anything from a cell phone to a production server. The idea is based on the availability of more and more components with differing computational power and levels of energy consumption. Additionally, these components have become cheaper. So why not put several components of varying power on the same device and switch between them as necessary? This can provide very smooth scaling.

Mike Schroeder (MSR) asked if this could be applied in datacenters. Mark said that is definitely possible and there are some projects that already try to do this. Peter Bailis (Harvard) asked about the programming model in such environments. Geoffrey said that there are ways to design convenient programming models for such systems. He gave an example of fat binaries that support several platforms. Prabal Dutta (U. Michigan) asked about the components that already include some method of scaling—for example, CPU frequency scaling. The authors replied that their design can reuse such features. One can switch to a more powerful CPU only if all levels in the currently working CPU are used up.

Mobile Apps: It's Time to Move Up to CondOS

David Chu, Aman Kansal, and Jie Liu, Microsoft Research Redmond; Feng Zhao, Microsoft Research Asia

David Chu noted two tendencies in mobile devices: (1) they are highly programmable and (2) more and more sensors are installed on these devices. As a result, programs that use sensors are becoming very widespread. Currently, they access sensors through inflexible custom interfaces. The approach the authors suggest is CondOS, an operating system that provides a unified interface for all sensors and applications. The OS will convert data to CDUs (Context Data Units) that are returned to applications. The benefit is that applications can perform a wider variety of tasks: for example, preload calendars when a user comes into the office or auto-unlock passwords when a user is at home.

Justin Pulaski (MIT) observed that the pervasive computing community has tried to do this for a long time already, but they are struggling to come up with a proper programming model. David replied that at the moment their interface is just a single syscall to get CDUs. Another person wondered why not employ user-level solutions such as the Linux D-BUS? David said that this should not be necessary with a kernel solution, but there should be some unified interface for all programs.

Free Lunch: Exploiting Renewable Energy for Computing

Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W. Moore, and Andy Hopper, Computer Laboratory, University of Cambridge

Sherif Akoush proposed moving computation and data processes to the places where green energy is generated. Governments may push industry toward being greener. Some companies have already installed solar panels near their datacenters. However, the amount of solar and wind power changes over time in a specific geographic region. If one can find two regions so that at least in one of them at any moment of time there is enough sun or wind to generate the required amount of energy, then one can migrate data and computation processes between corresponding datacenters dynamically. Migration can happen in the form of VM migration. The challenges one will have to address are storage synchronization, predictive VM migration, scheduling, and planning.

The authors did a case study in which they picked two datacenters, one in Africa and one in Australia. The downtime was only 0.5 seconds per migration, totaling 415 seconds per year, which corresponds to a very solid SLA. The cost of migration is 57.5 kJ/migration, which is also very low.

Aman Kansal (MSR) pointed out that a lot of hardware will be idling in this case. Sherif replied that energy will very soon

be more expensive than hardware. A lot of people in the audience did not believe that, saying that power should become really expensive in order for this technique to start to make sense. A related question was whether the authors neglected the cost evaluation. Sherif said that at the moment they do not have a good cost model. Another concern was high latencies. The author agreed that for some applications this approach will not work. Mike Freedman asked why they were using VM migrations. Sherif answered that for some application types, where the working set is small, VMs make sense.

Discussion/Open Mike

Summarized by Rik Farrow (rik@usenix.org)

The half hour of open discussion at first stayed focused on energy saving, the topic of the previous session. Dan Wallach wondered where else we could apply the genie. Geoff said that they had focused on improving the energy footprint of a single machine, but you could consider clusters and clouds. Mark Hempstead pointed out that the cost of transitioning processes or VMs between systems or DCs needed to be taken into account. Jeff Mogul mentioned that energy and computing is where computer security was ten years ago. Security is hard to get right, and accounting is the Achilles' heel of these things. The energy cost to produce a laptop is the same as the cost of using it two years. Mark responded that he hoped we would read his paper carefully, as they were careful. Geoff actually agreed with Jeff, in that we have been using voltage scaling for ten years, and Windows still does this so poorly it is better just to turn off the laptop. Mark mentioned that if we are really going to consider scalable computing, we need to consider the entire lifecycle. Mike Schecter said that people building datacenters are watching out for their own interests, but even they do not have control of all costs, including lifecycle costs.

John Ousterhout displayed a slide from RAMCloud (a Stanford project that replaces large disk storage with DRAM in server clusters). John pointed out that while disks have 16,667 times more capacity, latency has improved much less (twice as fast), while transfer rate is 50 times better than it was in the mid-'80s. But because capacity has far outstripped latency and bandwidth, reading an entire disk, using small blocks at random addresses, has become 8333 times worse. Just reading an entire disk sequentially can take 30 hours. Peter Honeyman said that the same thing is happening with memory, but John replied that memory is still much faster. Mike Swift noted that it is faster to read from a remote cache, outside the network, than to read from the local disk. The speed of doing computation over distance is the fundamental issue. Someone from Google pointed out that DCs are expected to last for 15–20 years, and using optimistically chosen costs are just going to get you laughed at.

Gernot Heiser mentioned that with DVFS it is very difficult to get even 10% power savings, as operating voltages have dropped to close to 1 volt. Gernot also pointed out that the Thumb instruction set in the ARM chip does not save energy. It is a subset of the regular ARM ISA, but you need to execute more instructions to get the same work done. You just get a smaller memory footprint.

Nobody Likes Surprises

Summarized by Suparna Bhattacharya (suparna@csa.iisc.ernet.in)

Debug Determinism: The Sweet Spot for Replay-Based Debugging

Cristian Zamfir, EPFL, Switzerland; Gautam Altekar, University of California, Berkeley; George Candea, EPFL, Switzerland; Ion Stoica, University of California, Berkeley

Replay-based debugging is a useful technique for tracking down hard to reproduce non-deterministic bugs which may otherwise take days or months to diagnose. The high runtime overhead involved in ensuring deterministic record-replay, however, is a major barrier to making these tools practical for production use.

Cristian Zamfir argued for a new model of determinism, called "debug determinism," which specifies that a system should at a minimum reproduce the failure and the root cause of the failure in order to be useful for debugging. Thus, debug determinism maximizes debugging utility, yet it is a relaxed-determinism model that has the potential to be achieved with low in-production overhead. He observed that existing relaxed deterministic replay approaches such as output determinism and failure determinism may end up sacrificing debugging utility in the process of reducing runtime overhead. For example, an output-deterministic system may only record the output but not the input context or data race that is the root cause of the failure. Debug determinism, on the other hand, relaxes determinism while ensuring that both the original failure and the root cause can be reproduced.

How might this be achieved? One could apply high-fidelity recording during portions of execution where root causes and failures are suspected—the key difficulty, of course, is that these are not known a priori. Hence, static analysis or domain knowledge is required to guess the location of possible root causes. In the case study presented for Hypertable, the authors relied on previous reports that control plane code tends to be responsible for most program failures, but only a small portion of the execution time. Thus, one approach is to record with high fidelity just the control plane. Cristian

proposed a metric called debugging fidelity (DF) to assess different approaches with respect to their debugging utility. For example, DF is 1 when both the failure and the original root cause can be reproduced (e.g., when both are in the control plane in the Hypertable example) and it is 1/3 when there are 3 possible root causes for a reproducible failure and the system may reproduce one of the root causes that is different from the original cause.

Jeff Mogul asked whether it would be useful to implement a two-phase approach involving a run in high performance (relaxed determinism) mode followed by other runs with high fidelity with respect to the possible root cause of the failure. Cristian pointed out that replay debugging systems are typically targeted at failures that occur infrequently and are hard to reproduce; therefore a two-phase approach may not work well in these cases. Mike Schroeder wondered whether low fidelity might be better, since it is good to know all the root causes, in order to fix them all. Cristian responded that finding all root causes for a failure may take a long time and may be more difficult to scale. However, such a system would have higher "debugging effectiveness," which is a different metric from debugging fidelity.

One participant asked for a clarification on how one can know up-front where the root causes are. Cristian replied that one could over-approximate where root causes are likely to be based on a heuristics or static analysis, then record those parts of the execution with high fidelity. For instance, one might be able to statically over-approximate where all the data races are. Some of these data races may be benign; due to the over-approximation, they would be recorded as well, yet the system would achieve debug determinism for failures caused by data race bugs.

Non-deterministic Parallelism Considered Useful

Derek G. Murray and Steven Hand, University of Cambridge Computer Laboratory

In contrast with much recent work that treats non-determinism as a source of undesirable problems in parallel programming, Derek Murray made a case for extending distributed execution engines to enable explicit support for non-deterministic execution in applications that can benefit from it. He began his presentation by explaining how distributed execution engines (e.g., MapReduce) take care of a lot of parallel programming drudgery, including parallelization, synchronization, scheduling, load balancing, communication, and fault tolerance. It is the last of these that requires deterministic execution. Thus non-determinism comes at the cost of trading off transparent fault tolerance. However, he argued that more efficient and versatile programs can be built if non-determinism is supported as a first-class abstraction in distributed computing engines, and that this could be achieved without forcing additional complexity on computations that do not involve non-determinism. He presented examples like branch-and-bound and applications with irregular-sized parallel sub-trees; these can be speeded up significantly by reducing wasted work, using primitives like asynchronous signals for work shedding and non-deterministic select to continue execution without synchronization delays.

Since the main challenge with implementing non-determinism lies in dealing with faults, Derek discussed a possible range of policies, from a conservative but expensive record and replay to explicit error/exception handling by applications, to the other extreme of a fail-everything all-or-nothing approach. One of the more interesting alternatives proposed was that of bounded non-deterministic annotations for computations that have deterministic outputs, but which may be implemented internally using non-deterministic steps for efficiency. The paper also describes how tainting could be used to differentiate non-deterministically generated outputs from deterministic references to restrict the impact.

Mike Schroeder asked about the extent to which the authors have managed to act on these observations. Derek responded that they need to understand the distribution of failures before concluding what the appropriate solution should look like. Cristian Zamfir wondered how one would deal with undesirable non-determinism such as a bug in the JVM or the kernel. Derek clarified that they were not trying to deal with those kinds of problems, but were focused on explicit user-level non-determinism. The main message here is that currently the problem of handling non-determinism has been pushed to lower layers of the system; instead, we should pull back some of it to higher layers where it may be cheaper to handle and enable more flexibility.

Finding Concurrency Errors in Sequential Code—OS-level, In-vivo Model Checking of Process Races

Oren Laadan, Chia-Che Tsai, Nicolas Viennot, Chris Blinn, Peter Senyao Du, Junfeng Yang, and Jason Nieh, Columbia University

This intriguing title marked the last talk of the session on non-determinism. Oren Laadan highlighted an important problem that has received very little attention in the systems community compared to the active research on thread races. This is the existence of process races, or races which occur when multiple processes access shared OS resources without proper synchronization, e.g., non-determinism in the results of ps aux | grep XYZ or a shutdown script unmounting a file system before another process writes its data. Using results from their survey of sampled race reports for common Linux distributions, he pointed out that process races are numerous and growing over the years. They can also be

dangerous, resulting in data loss and security vulnerabilities. Diagnosing process races is challenging, however, because of: (1) the diversity in scope (involving multiple programs written in different languages with complex interactions involving a variety of heterogeneous resources); (2) the need for a race detection algorithm that can handle these complex and often underspecified interactions between system calls and resources; (3) the difficulty of ensuring coverage due to dependencies on elusive conditions such as timing, environment configuration, and usage scenarios; and (4) a high likelihood of false positives or benign races.

Oren described their solution to the problem: RacePro, a system which combines lightweight online in-kernel record/replay (to transparently track access to shared resource accesses at the OS level) with an offline exploration engine that analyzes the record (using model checking) to detect potential process races. While the first piece addresses the scope challenge, the second addresses coverage. The algorithm challenge for race detection is solved by mapping this to an equivalent memory race detection problem which treats resources like memory locations and system calls like memory read/write. The last step of the solution is an offline validation using a live replay of a modified version of the recording that forces candidate race conditions to help rule out false positives. With their preliminary implementation they have detected 14 races, including 4 that result in a data loss, 5 that result in a crash, and 5 security vulnerabilities. Of all the races detected by the exploration engine, only 3–10% proved harmful, showing that the validation step is crucial.

There were several questions about what the underlying recording scheme actually captures and the assumptions made. Oren explained that they record all system calls and the partial order of their access to resources. Responding to a question from Marcos about whether they rely on a model of the OS for knowledge of what the shared resources are, Oren mentioned that their record replay mechanism is based on Scribe, their earlier work on a transparent lightweight application execution replay, published at SIGMETRICS '10. The basic resources are decided up front—e.g., IPC, files, inodes (not every single lock in the kernel), and partial ordering for a resource are recorded/effected by tracing internal kernel function accesses. Since Scribe can replay any application, including one that is multi-process and multi-threaded, RacePro can detect process races that involve threads as well. David Holland asked how robust the mechanism is to the kernel that's not working properly. The answer was that the approach assumes a correctly working kernel.

The Tin Foil Hat Session

Summarized by Srinath Setty (Srinath@cs.utexas.edu)

Privacy Revelations for Web and Mobile Apps

D. Wetherall and D. Choffnes, University of Washington; B. Greenstein, Intel Labs; S. Han and P. Hornyack, University of Washington; J. Jung, Intel Labs; S. Schechter, Microsoft Research; X. Wang, University of Washington

Right now, the research community's work can be divided into the following two categories: first, creating clever attacks to expose privacy risks, and second, devising narrow mechanisms to prevent a class of privacy risks. David Wetherall argued that the research community needs to go beyond these two classes of work and devise operating system mechanisms for privacy revelations. Privacy revelations will track how a user's information spreads in applications and will present that information to its users. The authors argue that this information will enable users to improve privacy if it can be presented as application-level concepts.

Yinglian Xie (MSR) pointed out that the problem is more than transparency: users need to know how their data gets used outside. Wetherall agreed. Matt Welsh from Google asked about the incentives for OS developers and application developers to support privacy revelations. Wetherall said that the work is not to disallow apps from tracking/collecting users' information but to expose that fact to the users. John Wilkes (Google) suggested that privacy revelations should go beyond what the authors defined: the operating systems should point out information about the ways in which the tracked information gets used. Wetherall agreed.

Do You Know Where Your Data Are? Secure Data Capsules for Deployable Data Protection

Petros Maniatis, Intel Labs Berkeley; Devdatta Akhawe, University of California, Berkeley; Kevin Fall, Intel Labs Berkeley; Elaine Shi, University of California, Berkeley and PARC; Dawn Song, University of California, Berkeley

Petros Maniatis began with a story about health data. His foot was injured in Palo Alto, and then he was hit by an ambulance, re-injuring the same foot, while in the UK. It would have been useful to have data from the medical work done in California while in the UK. But it is important to maintain control over our own health data.

Maniatis presented the secure data capsules vision: the owner of data sets a policy; policy is enforced during its lifetime, and data provenance is maintained throughout.

Then Maniatis presented the challenges involved in realizing this vision. First, the vision requires tracking information, which is known to be expensive in practice; devising efficient mechanisms to track the flow of information is a challenge. Second, the vision requires us to devise composable and meaningful policy definitions. Third, covert channels are a serious threat and need to be addressed.

Toby Murray (NICTA/UNSW) pointed out that microkernels are good for secure data capsules. Then Toby asked if naming is going to be the hard problem. Maniatis agreed but pointed out that it needs to be solved in order to realize the proposed vision. An audience member pointed out that Palladium at MSR, with a similar vision, had problems with displaying output on commodity hardware, and asked if this is going to be a problem in this work. Maniatis said there are solutions to the secure display problem if there is hardware manufacturer support. Michael Swift (University of Wisconsin) asked if it is going to be a problem to create policies to handle medical data before the data gets used. Maniatis pointed out that there are a couple of ways to handle this: the system could have policy violation budgeted to address the unknown data usage information, or a quorum of entities could decide the policy dynamically at runtime.

Making Programs Forget: Enforcing Lifetime for Sensitive Data

Jayanthkumar Kannan, Google Inc.; Gautam Altekar, University of California, Berkeley; Petros Maniatis and Byung-Gon Chun, Intel Labs Berkeley

Gautam Altekar explained that their idea is to create OS mechanisms to ensure that sensitive data is not retrievable after a defined data lifetime date has expired. This mechanism should not require support from applications. Altekar presented their initial work, state reincarnation, in which an operating system rolls back the application's state, replaces sensitive information with equivalent non-sensitive information, and rolls forward the application. State reincarnation eliminates any sensitive data from the system after its lifetime, but the challenge in achieving this is to derive equivalent non-sensitive data during the process. Altekar pointed out that output deterministic replay (SOSP '09) can be used to solve this problem in many cases, but overheads are going to be high. The talk also presented overheads by recording information at user-level: for bash, the slowdown was 1.2 times.

An audience member asked whether the goal could be achieved by simply going back in time. Altekar pointed out that that proposed fix would disrupt the application and the user. John Ousterhout from Stanford asked if this would increase risk by recording information. Altekar answered that users have to trust the recording system to not leak information. Timothy Roscoe from ETH Zurich asked if this is going to be used, since users may not like to record all their actions. The answer was to reduce the costs to make it favorable for the users to use it.

Discussion/Open Mike

No report is available for this session.

MacGyver Would Be Proud

Summarized by Sherif Akoush (sa497@cam.ac.uk)

Exploiting MISD Performance Opportunities in Multi-core Systems

Patrick G. Bridges, Donour Sizemore, and Scott Levy, University of New Mexico

Patrick Bridges presented opportunities to increase the speed of fixed-size workloads proportional to the processor count. He argued for strong scaling in systems software, and he gave the example of a single TCP connection as why we need it. For small MTUs, TCP synchronization across multiple cores is a bottleneck and it kills performance. Multiple-instruction/multiple-data (MIMD) approaches do not solve this problem, as they require coordinating activities between cores.

The alternative approach is to use a multiple-instruction/single-data (MISD) execution model based on the replication of sequential code across cores. In other words, synchronization is being replaced by replicating sequential work to guarantee consistency. They have implemented the system, and the initial result is that their model scales well for TCP receive processing. Patrick concluded by giving other examples, such as high-throughput file systems, in which the MISD approach would be beneficial.

Timothy Roscoe from ETH Zurich asked whether TCP is the interesting case in this approach and if it has any sequential component that can be replicated across cores to achieve scaling. Patrick answered that TCP state information such as window size, congestion control, and flow control is basically the sequential code that needs to be consistent across cores. He believes that TCP would be the killer application, to speed up a single connection flow proportional to the number of cores. Mike Swift asked about other applications that would fit this model. Patrick said that moving a large chunk of data in the DB world would be interesting as well.

More Intervention Now!

Moises Goldszmidt and Rebecca Isaacs, Microsoft Research

Moises Goldszmidt argues for what-if scenarios for data-parallel systems (e.g., MapReduce and Dryad). Unfortunately, this cannot be done by passive observations only; active interventions are required to learn the causality of different parts of the system. The proposed approach makes use of well-developed mathematical models, theories, and engineering.

The approach relies on passive observations to build the confounding factors that require further active interventions. Then, a Bayesian network is developed and executed which determines the experimentations needed. Statistics and machine-learning techniques provide a set of new rules that can be used for active interventions.

Matt Welsh said that he was the reviewer that said you are reinventing control theory, something that was done for 50 years. Can you apply stuff that was done 30 years ago? Moises answered that they actually combined different models to come up with a new formulation that can be used to decide on the active interventions. Rodrigo Fonseca (Brown) asked how the blueprint (i.e., causality) is captured, and if it is captured wrong, how this might affect the conclusions. Moises answered that the blueprint is constructed from passive observations (e.g., data sizes from nodes in the cluster that are running the tasks), while active interventions are what actually correct any wrong assumptions in the blueprint.

make world

Christopher Smowton and Steven Hand, University of Cambridge Computer Laboratory

Christopher Smowton argued that programs such as Firefox, OpenOffice, and Eclipse are rubbish since they repeat work that is done in either the current or the previous session. Developers of these programs never consider specialization of the software, because it is a challenging task. A spell-checker, for example, converts a local/global dictionary into a machine-readable form every time before checking. An easy optimization is to do this conversion only once per session. Alternatively, the program can be manually rewritten to save its intermediate results.

The paper proposes a more efficient technique based on global optimization: automated specialization of programs by partial evaluation. In the specific example of the spell-checker, this technique treats the dictionary as a constant propagated through the rest of the program. A prototype has been implemented as a proof of concept (not for the spell-checker) which can eliminate the redundant work of counting words in a file. However, there are challenges in making this adaptive optimization efficient and between multiple processes.

Phil Levis (Stanford) asked how this adaptive optimization would work for specialized paths in the current user's home directory. Christopher answered that this can be mitigated by either pushing this challenge to the user or writing a wrapper script that can check whether there is a specialization version available. Petros Maniatis (Intel Labs Berkeley) asked how this approach compares to speculative execution. Christopher answered that what he is proposing is complementary, as it can eliminate some decisions that are not needed ahead of the speculative execution. An audience member commented that the proposed approach can be augmented to model deferential execution of multiple program invocations over time to identify common state which is useful. Dave Holland (Harvard) wondered about what happens if the content of the file changes during execution. The reply was that, for the time being, it will be left to the developer to take the correct action.

Poster Session

Summarized by Lon Ingram (lawnsea@cs.utexas.edu)

The Best of Both Worlds with On-Demand Virtualization

Thawan Kooburat and Michael Swift, University of Wisconsin—Madison

Kooburat and Swift propose on-demand virtualization, where users run natively most of the time to reap the full performance of their hardware, but can switch to running in a virtual machine when needed. They save the state of the OS and running processes through hibernation, use hotplugging to transition devices from physical to virtual hardware, and employ logical devices to preserve device bindings, which allows network connections to be maintained.

Mobile Apps: It's Time to Move Up to CondOS

David Chu, Aman Kansal, and Jie Liu, Microsoft Research Redmond; Feng Zhao, Microsoft Research Asia

Chu presented his team's vision for a new kind of mobile OS service. The OS would provide applications with context signals—whether the user is standing or sitting, for example, or in a loud or quiet environment—in addition to raw sensor data. They claim that such an OS would better protect the user's privacy and use resources more efficiently, among other advantages.

Seeking Efficient Data-Intensive Computing

Elie Krevat and Tomer Shiran, Carnegie Mellon University; Eric A. Anderson, Joseph Tucek, and Jay J. Wylie, HP Labs; Gregory R. Ganger, Carnegie Mellon University

Krevat and his team investigated what inefficiencies affect data-intensive scientific computing (DISC), which they define as large-scale computations over big datasets. They used a simple model of DISC and a library called Parallel DataSeries to look for performance problems.

Detern: Robustly and Efficiently Determinizing Threads

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

Yang and his team built on recent work in the field of deterministic multithreading, making it more robust and efficient. Their system caches thread schedules and reuses them; it also uses a hybrid schedule that takes advantage of the fact that there are typically relatively few races during the execution of the program.

Execution Synthesis: A Technique for Automated Software Debugging

Cristian Zamfir and George Candea, Ecole Fédérale de Lausanne

Zamfir and Candea created a method for automatically finding a path through a program that reproduces a reported bug. Their technique uses a focused path search based on a combination of heuristics and symbolic execution to reach a target failure state. It incurs no runtime overhead and can find deadlocks and race conditions.

Memento: In-Memory Caching for Datacenters

Ganesh Ananthanarayanan, Ali Ghodsi, and Andrew Wang, University of California, Berkeley; Dhruba Borthakur, Facebook; Srikanth Kandula, Microsoft Research; Scott Shenker and Ion Stoica, University of California, Berkeley

The Memento team are working on a memory cache for data-intensive workloads in datacenters. They observed that most jobs are small and require all of their data to be cached to reap performance benefits. Large jobs, on the other hand, experience linear improvement as their working set is cached. Traditional caching disciplines ignore the all-or-nothing constraint on small jobs. Memento categorizes jobs by size and tries to ensure that this constraint is met so that large jobs don't starve small ones.

Pervasive Detection of Process Races in Deployed Systems

Oren Laadan, Chia-Che Tsai, Nicolas Viennot, Chris Blinn, Peter Senyao Du, and Junfeng Yang, Columbia University

This project uses a recording of process interactions through the system call interface to detect process races. Once a race is detected, the system re-executes the processes until immediately before the racing syscalls. It then resumes execution, but with the loser of the race executing before the winner. This technique allows them to reduce false positives by ignoring benign races.

Sirikata: Design and Implementation of a Next Generation Metaverse

Philip Levis, Stanford University; Michael J. Freedman, Princeton University; Ewen Cheslack-Postava, Daniel Reiter Horn, Behram F.T. Mistree, and Tahir Azim, Stanford University; Jeff Terrace, Princeton University; Bhupesh Chandra, Stanford University; Xiaozhou Li, Princeton University

Sirikata is a project to actually build a usable virtual world. The challenges presented by such an undertaking are unique and daunting. The project is well into its implementation, with 12 undergraduate students building a virtual city called Merustadt over the summer of 2011.

SPECTRE: Speculation to Hide Communication Latency

J.P. Martin, C. Rossbach, and M. Isard, Microsoft Research SVC

Programs that share mutable state and sequential algorithms are harder to run efficiently on multiple machines. SPECTRE uses prefetching and speculative execution to run sequential algorithms in parallel, rolling back if a conflict is encountered. It is currently running as a prototype on a small cluster.

T2M: Converting I/O Traces to Workload Models

Vasily Tarasov, Santhosh Kumar Koundinya, and Erez Zadok, Stony Brook University; Geoff Kuenning, Harvey Mudd College

T2M is an effort to create benchmarks from I/O traces. Traces are broken into chunks based on what model will be used to analyze the chunks. The chunks are then modeled and a workload model is output, which can be used as a benchmark in the future.

Why a Vector OS Is a Bad Idea

Vijay Vasudevan and David Andersen, Carnegie Mellon University; Michael Kaminsky, Intel Labs

Awarded Best Poster!

Vasudevan presented the winning poster, which discussed the downsides to the Vector OS project that he discussed in the final session of the workshop. Two problems identified on the poster were the difficulty of programming to an explicit vector interface, illustrated with code showing the extra work required, and latency penalties paid when code diverges. The poster also included a space for audience members to fill in their own objections to the scheme.

Wild and Crazy Ideas Session

Summarized by Thawan Kooburat (kooburat@cs.wisc.edu)

Matt Welsh (Google) presented MEME OS, which is designed to appeal to the Internet generation. This generation does not understand the messages displayed via the text-based terminal. He proposed the use of funny images from the Web as a way to report output or error messages to users.

Margo Seltzer (Harvard) conducted a poll on how people carry their mobile phones. She found that those who carry their phones in their pockets are mostly men. This means that women may not carry the phone with them when leaving their desk to do small errands. Research on mobile phones should also take women's behaviors into account, since they represent the other half of the demographic.

Jeffrey Mogul (HP Labs) proposed a journal for reproduced results in OS research. He also came up with several ideas to provide incentive for people to work on this journal. For example, submitting a paper to this journal should be required to get tenure. The authors of any system paper need to put down $1000 on paper submission, which will go to the reviewers if the result is refuted within two years. Many people responded that other research communities—the database community, for example—already have mechanisms such as reproducibility committees to verify published works.

Dan Wallach (Rice University) complained about the current submission process, in which papers get into the loop of submit-reject-revise. He proposed that all papers should get accepted immediately as tech reports. This sparked a debate where people discussed the submission process of other conferences such as SIGMOD and VLDB. Others also raised the idea of removing anonymous review or using crowdsourcing instead of peer review.

Mike Walfish (University of Texas at Austin) showed a YouTube video where a penguin is taught to go shopping. He suggested that robots should be used to perform mundane tasks like this. He presented his work in building robots which are easy to program and able to perform simple tasks such as getting a cup of coffee. He proposed a model where people can download pre-programmed tasks to their robots from places like AppStore.

David Anderson (CMU) suggested that systems research is about dealing with constraints imposed by hardware. Previously, we have been able to use many abstractions to hide some hardware details such as uniform memory and sequential computation. However, as we are hitting the physical limit of physical devices, we will start to throw away these abstractions. He believed that we will ultimately reach the end of scaling of physical devices and we should accept this fact. Because of this, we should think carefully about which abstractions to throw away and in which order, so that programmers will continue to survive despite these changes.

Joseph Tucek (HP) explained that the next-generation computer system is just a machine with a different ratio of hardware resources. He proposed that cutting-edge research can be carried out by putting together machines that simulate this ratio. For example, we can pair a 386 processor with a 10 Gbps network to mimic the future Terabit network.

Prove It!

Summarized by Sherif Akoush (sa497@cam.ac.uk)

What If You Could Actually Trust Your Kernel?

Gernot Heiser, Leonid Ryzhyk, Michael von Tessin, and Aleksander Budzynowski, NICTA and University of New South Wales

Gernot Heiser presented an seL4 microkernel that is formally proven to be functionally correct. The kernel is free from crashes, bugs, and similar safety issues. Interesting applications are for the purposes of better virtual machine monitors and isolating Web browsers. Trusted platform modules (TPM) can also be made practical by the use of a trusted kernel with a trusted verified loader.

Taking home banking as an example, TPM is practically useless, as it forces the users to boot into a special banking configuration that will kill any other concurrent access to other machine features. Late launch/DRTM is also practically useless, as it does not allow for interrupts, DMA, or multiprocessing. The proposed solution is to load the banking application in a mini OS that is also loaded with a verified loader on top of a verified seL4 kernel. The user's standard OS is still working in parallel and is not affected. Additionally, DBMS would not need synchronous log writes, as it is guaranteed

with a verified kernel that the OS will not crash. In this case, there is no tradeoff between performance and reliability.

John Ousterhout (Stanford) asked if they had found any issues in seL4 since the SOSP '09 paper. Gernot replied that they had found a few proof bugs, around specification, configuration, and initialization. The only way around that is to complete the proof chain for the security parts. Brad Chen (Google) asked whether there is more than isolation that can be gained by a verified kernel and how application correctness can be guaranteed. Gernot answered that the guaranteed kernel functionality can be leveraged to ensure user-level component interfaces and this is something they are currently working on. Mike Swift asked what happens if the memory fails, and Gernot replied that people trust their RAID systems today. He also said that the military would like to have triply redundant memory for some applications.

Provable Security: How Feasible Is It?

Gerwin Klein, Toby Murray, Peter Gammie, Thomas Sewell, and Simon Winwood, NICTA and University of New South Wales

Toby Murray argued that provable security for a real system is feasible but certainly not easy. Real proofs are done by machines and can provide you with unexpected insights into high-level security issues such as integrity and confidentiality. Real systems are big and often written in C or assembler, not in a language that is designed to be proofed.

Toby provided seL4 as an example of a proofed kernel that enforces integrity. It is a machine-checked proof with 10,000 lines of proof-script code. However, timing channels are still too hard to be proofed and require a very detailed model of the underlying hardware. Additionally, systems like Linux cannot be proofed easily, as they have large trusted components.

Steven Hand (University of Cambridge) asked what is required if changes are made to the kernel. Toby answered that it depends on the level of modification done to the kernel; the correctness proofs rely on a number of invariants that have been proved about the kernel, and most of the work in proving correctness involves proving these invariants. So changes that do not break the invariants or introduce new ones require little work; however, ones that do require more work.

John Ousterhout (Stanford) asked about the number of lines of code seL4 has and how the effort required for the proof scales with the number of lines. Toby replied that seL4 has 8,600 LOC and noted that according to his experience, the proof should scale more than linearly (about square) with the size. Brad Chen (Google) asked how important the missing specification for hardware is. Toby answered that it is really important and they are actually working on modeling the hardware.

Toward Practical and Unconditional Verification of Remote Computations

Srinath Setty, Andrew J. Blumberg, and Michael Walfish, The University of Texas at Austin

Srinath Setty presented a practical and unconditional verification of remote computations which is useful in cloud and volunteer computing. Basically, the client needs to verify that the server has executed the code correctly without redoing the computation. One solution is to use probabilistically checkable proofs (PCPs), but PCPs are currently just applied in theory. The purpose of this paper is to make them practically possible (i.e., position the challenge as a systems problem). They refined PCP via arithmetic circuits instead of Boolean circuits, to make the system efficient, and implemented the design to demonstrate its practicality.

The prototype achieves savings of 10 orders of magnitude by using the refinements presented. It is based on a divide and conquer strategy: dividing the problem into smaller parallel parts that the server checks simultaneously and then verifies. However, more refinements are still required to reduce the storage cost and support floating-point operations, for example.

Steven Hand (University of Cambridge) asked why we should use PCPs instead of replication, as replication is simpler to verify computations. Srinath replied that replication assumes a threshold on the number of faulty servers, but PCPs provide stronger guarantees. Mike Freedman (Princeton) asked whether the optimizations that have been made can be generalized to the implementation of other computations. The answer was that these optimizations can be applied to any computation expressed as a circuit. In principle, a compiler can be used to translate a circuit representation from a high-level specification.

MOMMIE Knows Best: Systematic Optimizations for Verifiable Distributed Algorithms

Petros Maniatis, Intel Labs Berkeley; Michael Dietz, Rice University; Charalampos Papamanthou, Brown University

Petros Maniatis argued for an approach that guarantees the development of both algorithmic logic (for verification) and optimizations (for efficient implementation). Abstractions are great, but developers usually end up modifying the actual implementation code when systems are built. The actual code is therefore too difficult to be verified for correctness. Moreover, we should not worry about this level of implementation detail.

The proposed middleware, MOMMIE, is a high-level language that can be used to compose the system. This abstraction can then be translated to a specification (TLA+) for formal verification or to an optimized program (C++) for execution. In this way, proofs carry over into implementation without having to rebuild everything from scratch if the system changes.

A MOMMIE statement is the fundamental building block and is composed of an issuer, a verifier, an auditor, and C-structs. The program looks like event-condition-action, where actions have assignment, loops, and variables (i.e., imperative code). A prototype is available but it is still in its early stages.

Mike Freedman (Princeton) asked whether the designer tells MOMMIE which parts of the algorithm should go to the formal proof and which parts are for actual optimizations. Petros answered that they focused on abstraction so that anything that is composable can be proved in isolation and some aspects are mapped manually to implementation detail. Toby Murray asked whether there are any restrictions on the algorithm formulated in MOMMIE. Petros replied that there are some restrictions: for example, it cannot allow for arbitrary loops, because the goal is to reduce the amount of work a designer has to do. Others wondered how this work differs from other systems and protocols. Petros replied that MOMMIE provides a granularity (middleware) that is not found in any other system.

OS Design Isn't Dead; It's Just the Last Session of the Workshop

Summarized by Thawan Kooburat (kooburat@cs.wisc.edu)

The Case for VOS: The Vector Operating System

Vijay Vasudevan and David G. Andersen, Carnegie Mellon University; Michael Kaminsky, Intel Labs

Vijay raised the fact that in a Web server each request often executes a similar sequence of operations, such as accepting a network connection, opening a file, etc. Thus, a lot of redundant and identical work is performed when servicing requests in parallel using multiple cores. He proposed a vector system call as a mechanism to increase system efficiency. Vectorization allows batching of system calls, which reduces kernel crossing overhead, but, more importantly, it allows redundant work to be eliminated. Examples include reducing pathname resolutions, using SSE instructions in hash calculations, and looking up data structures more efficiently.

Vijay demonstrated the benefits of a vector system call while performing memory protection at the speed of millions of operations per second. This was achieved by vectorizing the mprotect system call. First, vectorization batches up several system calls into one, similar to FlexSC. Then it eliminates redundant TLB flushes and algorithmically exploits vector abstractions by sorting requests based on page address, which reduces memory allocation overhead. These optimizations provided a factor-of-three improvement in the mprotect rate: 30% of the improvement was attributed to avoiding system call overhead, while the other 70% came from the vector opportunities deeper in the stack, emphasizing the need for vectorization at all levels. Next, Vijay described the challenge of dealing with divergent execution paths. He proposed a solution based on either forking extra threads and rejoining them when execution paths converge, or using lightweight message passing between function calls. However, deciding when to fork and join execution is application-specific and remains a challenge. Finally, he described one way of building a vector OS by restructuring the OS as a staged event system. This allows the programmer to write sequential code and let the system handle vectorization.

Erez Zadok (Stony Brook) asked about the difficulty of dealing with errors when using vector system calls. Vijay replied that this task can be simplified by using an event-based model, as it allows programmers to handle errors individually. Andrew Baumann (Microsoft) pointed out that other OS designs explicitly avoided synchronization overhead by executing work redundantly in parallel. Vijay responded that eliminating redundancy at the cost of serialization improves efficiency, especially for I/O-bound operations in a highly parallel Web server. Mike Schroeder (Microsoft) brought up concerns about latency, since the system may have to wait to batch up requests. Vijay said that existing techniques such as interrupt coalescing already batch up requests at the network layer before they arrive at the application, providing an opportunity for vector execution to improve efficiency without adding significantly more latency.

Operating Systems Must Support GPU Abstractions

Christopher J. Rossbach and Jon Currey, Microsoft Research; Emmett Witchel, The University of Texas at Austin

Christopher raised the fact that the GPU, as a general-purpose computing device, is underutilized because it is treated as an I/O device. He strengthened his argument by showing that the CPU has a much richer set of abstractions, such as process and pipe, than the GPU, which has only ioctl as the main interface. He described several issues caused by the lack of a proper abstraction. First, there is no fairness or isolation guarantee from the kernel. Second, the absence of a kernel-facing interface means the kernel cannot use GPU directly. Third, he presented two experiments to highlight CPU/GPU performance isolation and scheduling problems.

Fourth, Christopher talked about his gestural interface program. He tried to decompose the program into a collection of programs connecting via pipes. However, this design has poor performance as a result of the unnecessary data movement. With existing GPU abstractions, this overhead cannot be removed.

Christopher emphasized that general-purpose GPUs need more abstraction, similar to what the CPU has. It needs many APIs to support functions such as scheduling and inter-process communication. The right abstraction should enable program composition and eliminate unnecessary data movement between CPU and GPU. The proposed abstraction is based on a dataflow programming model. First, PTasks represent a computation executing on a GPU. They have priority to allow the kernel to enforce fairness. They are also connected via ports and channels. These specialized channels allow programmers to eliminate unnecessary data movement when an opportunity arises. Finally, he revisited his gestural interface program to show how these abstractions solve the problem.

Erez Zadok (Stony Brook) suggested that if the GPU is incorporated into the CPU like the floating-point coprocessor was, this problem may go away. Christopher responded that having better hardware is also one of the solutions, but he would like to have a solution that works with existing hardware. Philip Levis (Stanford) argued that this problem is irrelevant, as the CPU is moving to multicore and becoming more heterogeneous. Christopher explained that dataflow is still important since it may be the right model for any type of accelerator. Brad Chen (Google) brought up concerns regarding GPU security issues and believed that it should not be tightly integrated with the OS. Christopher replied that better support from hardware, such as allowing context switching and better specification, can mitigate the problem.

Multicore OSes: Looking Forward from 1991, er, 2011

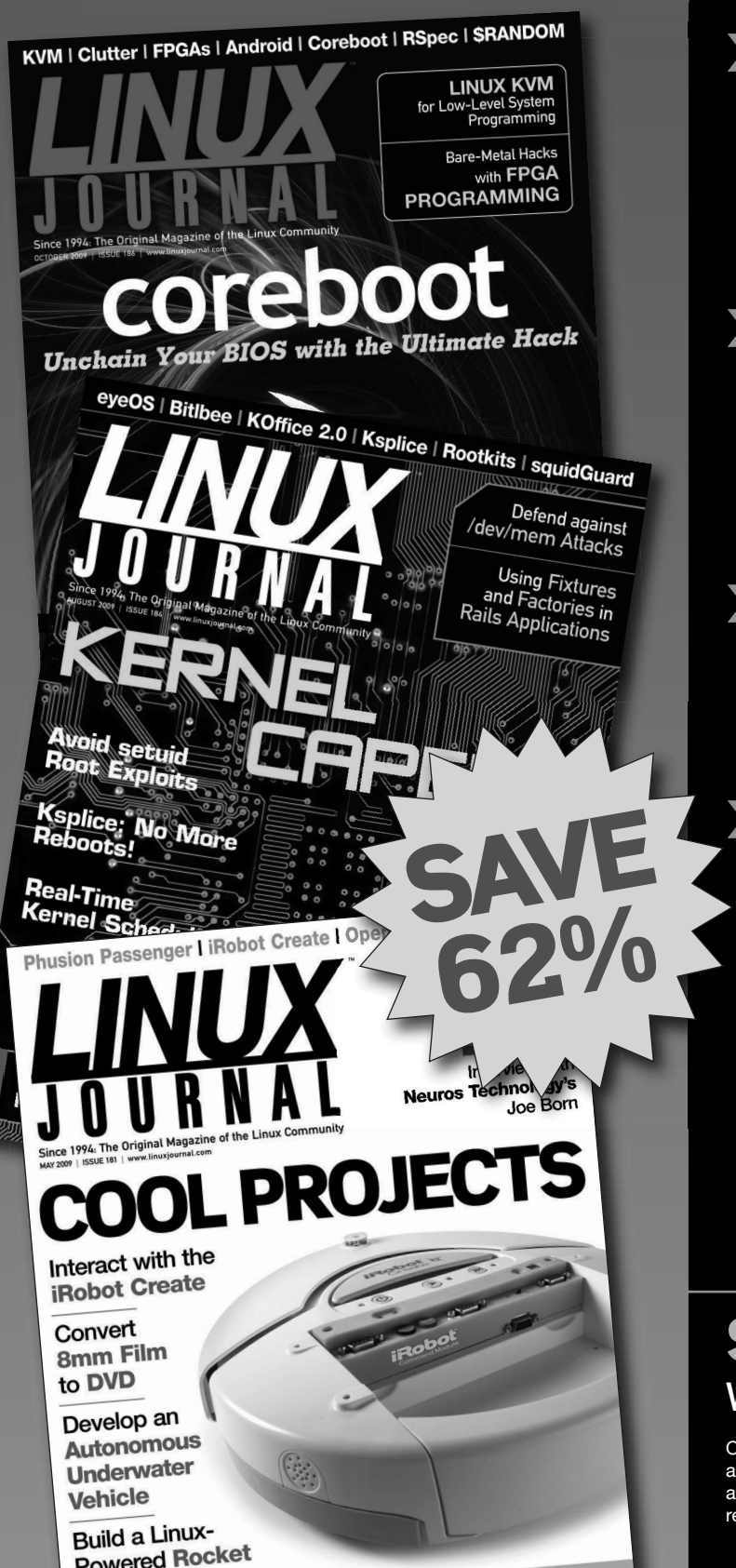
David A. Holland and Margo I. Seltzer, Harvard University

David complained about multicore systems. Hardware is not getting faster and we are forced to adopt parallel programming, which makes good scalability very difficult to achieve. However, these are the same challenges that people who worked with supercomputers faced around 1991. The experience gained from this shows that machines with thousands of cores will need to adopt the shared-nothing architecture. We also learned that message passing is the right model for programming these machines. However, instead of using MPI, a lightweight message channel is already available in languages such as Go and Erlang. Since it is based on a shared-nothing architecture, it has the potential to achieve good scalability. Unlike sending a MPI and network RPC packet, sending a lightweight message is comparable to a procedure call.

David talked about how to restructure the kernel to use messages. He presented a diagram which shows the kernel running on a separate core instead of underneath the application. In shared-nothing architecture this is possible, since there is no need to protect processes from overwriting each other. He also discussed several challenges in building such a system. For example, relying on a hardware-based channel can lead to a similar issue that people relying on Infiniband encountered. Virtual memory may also look entirely different, since there is no kernel running underneath. In addition, this model may encourage programmers to write too many threads. Finally, fault-tolerance and scheduling may become issues as well.

Timothy Roscoe (ETH Zurich) argued that some form of kernel is still required to run together with the application in order to perform privileged tasks on its behalf. Hence, only OS services are needed to be restructured around message passing and run on a different core. David responded that hardware-based channels can remove the need for running the kernel in the same core. Joseph Tucek (HP) raised the point that a NUMA machine with 1000 cores is already available today from SGI and is used by NASA. David Andersen (CMU) commented that this system tries to achieve Mach-like message passing with Go-like language support. He also mentioned that people usually wrap an RPC-like interface around message passing. David confirmed the point about language support and argued that a lightweight message can have as low overhead as a function call. On the other hand, RPC is too heavyweight to replace every function call in a program.

If You Use Linux, You Should Be Reading LINUX JOURNAL™



SAVE 62%

- In-depth information providing a full 360-degree look at featured topics relating to Linux
- Tools, tips and tricks you will use today as well as relevant information for the future
- Advice and inspiration for getting the most out of your Linux system
- Instructional how-tos will save you time and money

Get *Linux Journal* delivered to your door monthly for 1 year for only $29.50! Plus, you will receive a free gift with your subscription.

SUBSCRIBE NOW AT:

WWW.LINUXJOURNAL.COM/SUBSCRIBE

Offer valid in US only. Newsstand price per issue is $5.99 USD; Canada/Mexico annual price is $39.50 USD; International annual price is $69.50. Free gift valued at $5.99. Prepaid in US funds. First issue will arrive in 4-6 weeks. Sign up for, renew, or manage your subscription on-line, www.linuxjournal.com/subscribe.