;login:
OCTOBER 2013 VOL. 38, NO. 5
ELECTRONIC SUPPLEMENT

Conference Reports

HotOS XIV: 14th Workshop on Hot Topics in Operating Systems

Santa Ana Pueblo, NM
May 13-15, 2013

HotOS XIV Opening Remarks

Summarized by Rik Farrow (rik@usenix.org)

Petros Maniatis, Intel Labs, the PC chair, explained the ground rules for the HotOS '13. Presenters had only 10 minutes, with a few minutes for questions and answers as the next presenter set up his or her laptop. Each talk session was followed by a half-hour open mike session, where participants were welcome to speak on any topic, although the discussions were generally related to ideas brought up during the previous session or earlier in the workshop.

Petros also introduced a new concept: unconference sessions. Four sessions were set aside for groups to meet about topics of their own choosing. Attendees announced topics during a session on Monday morning and gave reports on the issues, and sometimes on the results of these meetings, on Wednesday, right before the end of the workshop.

Shuffling I/O Up and Down the Stack

Summarized by Shriram Rajagopalan (rshriram@cs.ubc.ca)

We Need to Talk About NICs

Pravin Shinde, Antoine Kaufmann, Timothy Roscoe, and Stefan Kaestle, Systems Group, ETH Zurich

Timothy Roscoe began by pointing out that modern NICs have become complex devices with a variegated set of features, but operating systems do not provide proper abstractions to access many of these features. Windows provides different abstractions for each NIC manufacturer, whereas Linux does not provide any support to access the hardware functionalities in modern NICs. Most operating systems as of now cannot optimize performance of a workload by automatically identifying and leveraging functionalities exposed by the NIC hardware.

Dragonet presents a new network stack design that represents the protocol state machine in the OS as a dataflow graph. The NIC's capabilities are represented as a dataflow graph as well. The two graphs can be combined in such a way that functionalities not provided by the NIC hardware can be provided by software components in the network stack.

Someone pointed out that graphics folks have taken a similar approach, and asked whether Mothy could draw a parallel between the two approaches. Mothy replied that their approach has a similar flavor; however, graphics cards are heterogeneous and provide arbitrary multiprocessing capabilities apart from functionality offload. His team is dealing with fixed function hardware. Someone else asked how high should the abstractions go up the stack: for example, the ability to push computations onto the NICs for certain workloads (e.g., receiver side scaling). Mothy answered that they don't know yet, but that they'd like to be able to offload processing to the NIC, but they need to track the spatial placement of threads. Brad Karp (University College, London) asked whether it is possible to automatically capture the NIC's capabilities in a protocol graph, when its firmware is updated, and if so wouldn't they have to update the OS's protocol graph accordingly. Mothy responded that you could treat this issue like a bug fix for bad firmware in the card. Until the firmware is fixed, the OS could use a different resource graph as a workaround. Their design just makes it easy to work around these hardware issues.

The NIC Is the Hypervisor: Bare-Metal Guests in IaaS Clouds

Jeffrey C. Mogul, Jayaram Mudigonda, Jose Renato Santos, and Yoshio Turner, HP Labs

Jeff Mogul started with a question: Why would anyone want to run a bare metal guest without a hypervisor? There could be several motivations, such as performance, security, application/vendor support for certain software, licensing requirements, and customer demand. The next question that naturally arises is how can one run both bare metal guests (BMGs) and virtual machines in the same cloud? With BMGs, we no longer have a guest OS running over a hypervisor, so where will the protection boundary be drawn? Jeff suggested using the Switch/NIC to enforce a hypervisor-like protection boundary for BMGs.

A simple inventory shows that we have several components already in place. For example, a sNICh provides ACLs with hardware NICs. Remote management can be accomplished via components such as HP's iLO (or equivalents from other vendors, using IPMI) with little modification. The Remote Management Engine (RME) at the end host interacts with the cloud controller. Depending on the requirements of the BMG, the RME configures the NIC with appropriate protection boundaries by disabling certain features; however, other things, such as checkpointing, migration, etc., require guest OS support. Jeff suggested that using an SDN is not the appropriate solution because BMG-NICs present a cleaner separation between the edge hardware and the network fabric and scales better.

Someone asked whether customers who demand bare-metal guests have concerns with licensing fees. Jeff answered that some applications cannot run on a VM, and apps would not be able to tell they were running over a sNICh. Muli Ben-Yehuda (Technion) asked whether this would still be necessary if the hypervisor had no performance penalty. Jeff pointed out that performance is just one aspect. A key driving factor for BMG-NICs is licensing and support requirements. Someone asked about the problem with RMEs accessing the main memory, and Jeff replied that because of their design (the BMC interface used by IPMI) RMEs do not

have a main memory map. Another person asked why the RME is even relevant. Jeff said that they need someone to control the NIC. Current systems allow RME to control the NIC. Basically, we are leveraging something that's readily available.

Virtualize Storage, Not Disks

William Jannen, Chia-che Tsai, and Donald E. Porter, Stony Brook University

Bill Jannen stated that virtualization works great because of hardware emulation but has a big performance impact on storage. For example, we have duplicated storage stacks in both the guest and the host—things such as page caches, read ahead blocks, etc.—when using a file-based backing disk. The double caching can cause correctness problems with certain file system operations in the event of failure. Bill described an example scenario where the guest issues an unlink system call on a file and gets an acknowledgement from the host; however, at the host level, the inode information still resides in the page-cache. Should the host fail and come back up, the guest OS's application would see the deleted file and might react in an undefined manner.

They proposed separating the media access layer from the file system. The application interfaces would reside in the guest while things like I/O schedulers would be at the host. They could then augment the guest API with performance, ordering hints, etc.

Steve Niel (VMware) claimed that VMware ESX servers do not have this issue; however, he appreciated the idea that we need to modularize the storage layer. Ed Yang (Stanford) said that this also applies to Xen and KVM, and that their example pertains to the configuration settings for their guest OS. Muli Ben-Yehuda said that the idea of modularizing certain aspects of storage, such as file systems, depends totally on the data structures that the file system uses. The case may be that such modularization is not possible for a given file system due to the nature of its data structures.

Unified High-Performance I/O: One Stack to Rule Them All

Animesh Trivedi, Patrick Stuedi, Bernard Metzler, and Roman Pletka, IBM Research Zurich; Blake G. Fitch, IBM Research; Thomas R. Gross, ETH Zurich

Animesh Trivedi stated that I/O performance has changed over the years. We have moved from disks to flash and will move to PCM, which represents two to five orders of magnitude performance improvement; however, the OS is not leveraging these features. We need a set of rich I/O semantics with direct access to hardware.

High performance I/O stacks work great with disks but don't perform well with NVRAMs. Instead of reinventing the wheel, he suggested, let's leverage the technology available in the networking community. Inspired by high performance software-controlled NICs, he proposed user-space mapped I/O channels with no OS involvement. An even better alternative would be to unify both I/O stacks. The OS could support a single set of abstractions for multiple sets of devices. The application would no longer care whether the storage is local or remote. Animesh said they have a working prototype that performs two to five times better with about a half million IOPS.

Muli Ben-Yehuda disagreed with Animesh's claim that network performance issues with respect to application access have been fully solved. Animesh replied that they do not claim that it's fully solved. Their opinion is that certain aspects of this space have been fully fleshed out and they propose to leverage them. For example, the OS would do a one-time translation to set up the I/O channel, acting like a control plane, for a very large file transfer. John Ousterhout (Stanford) asked what if there were a very large number of small files, which would be doing too many checks and hurting latency. Animesh agreed that too many data/control plane switches would have an impact on performance. Ed Bugnion (EPFL) pointed out that in networks, the socket is the central abstraction. In storage, its equivalent is SCSI. Their example is to use a niche network example (direct hardware access) and build a system on top of it. So at best, it's a niche within a niche. Animesh countered that sockets don't do high-speed transfers of hundreds of GBs of data. If you need high performance I/O, you need a niche. Simon Peter (U Washington) asked, what if two applications want to access the same file? Animesh said that you just remap the same channels with multiple processors and assume that the hardware can keep track of the ordering. Andrew Warfield pointed out that Animesh had focused on the similarities between the two domains, and asked that Animesh provide a big difference that is challenging. Animesh replied that networks have no notion of transactions while storage uses a lot of transactions. We have no way to roll back a transaction when doing I/O over network (but we can over storage).

Open Mike

Matt Welsh (Google) asked whether we know the kind of applications that are driving the kinds of papers that were seen in the I/O session. Do all applications need these features, such as direct access to I/O, or is it just a few? Timothy Roscoe responded that trading applications is a good use case because they cannot afford the hit on latency. He agreed that the customer base was a small one and that the application domain for these ideas was small.

Jeff Mogul commented that HPC applications are difficult to manage as they grow—especially resources, I/O, etc. The concepts presented in the session basically proposed abstractions that help the application/user easily manage these resources. Alex Snoeren (UC San Diego) added that, although these papers proposed to take the hardware capabilities to user space, hardware vendors (e.g., storage) are moving in the other direction (keeping to kernel space) in an effort to be compatible with each other. They don't want user-space libraries directly accessing their devices and creating compatibility issues.

Muli Ben-Yehuda reiterated Alex's observation that vendors are trying to move interfaces to the kernel because of legacy applica-

tions. He added that a major issue with direct hardware access is the loss of ability to migrate VMs and cited SR-IOV as one example. For enterprises with legacy applications, migration is a valuable tool compared to direct hardware access. Dave Ackley (U New Mexico) pointed out that NICs are getting smarter; it's the manifest destiny of silicon. Just as GPUs have been growing in capabilities by leaps and bounds, expect the same thing to happen with network processing. George Candea (EPFL) wondered whether a coordinated hardware/software design is needed to get the desired performance. The current approach is a real hodgepodge. Andrew Warfield (UBC) said that the current network stack is a real mess, with 15 vendors and only two of them focused on performance. Muli reiterated that moving code into user space wouldn't work for legacy applications. Steve Hand (Cambridge and MSR) said that once you bypass the hypervisor, you can no longer migrate, and people like the ability to do migration. So is this what customers really want?

Petros summarized by saying that this is a puzzle with multiple sides. Being able to mix-and-match and optimize for a particular solution would be nice. All sides have a point here—splitting things into small pieces, pushing some into hardware.

Edgy at the Edge

Summarized by Jonas Wagner (jonas.wagner@epfl.ch)

The Case for Onloading Continuous High-Datarate Perception to the Phone

Seungyeop Han, University of Washington; Matthai Philipose, Microsoft Research

Seungyeop Han introduced the case for onloading continuous high-data-rate perception onto the phone by explaining how computer vision has reached maturity and enables many applications, from context-sensitive reminders to tracking the user's diet. To perform sensing on the phone for continuous availability, cost, and privacy is desirable. Trends in memory size, processor speed, and power consumption indicate that this will be feasible in 2015.

A key optimization for on-phone video processing is using other sensors to gate the computation. These sensors identify frames that need not be processed, e.g., due to low light or motion blur, and discard more than 98% of all frames. This gating framework, combined with privacy concerns and the possibility to share models and algorithms between apps, calls for implementing video processing as an operating systems service.

Vova Kuznetsov (EPFL) asked whether gating is still useful if interesting frames come in batches. For many applications, gating still provides considerable energy savings. Matt Welsh (Google) asked whether this is really an OS problem. Seungyeop replied that techniques such as gating require multiple resources to be scheduled and shared between apps. Also, the OS can ensure privacy in the presence of malicious apps. To a follow-up question on privacy, Seungyeop replied that there are further ideas: for example, filtering an audio frame such that it is possible to identify the speaker but not the content. When asked whether his work makes offloading obsolete, Seungyeop said that, although some classes of applications require the cloud for reasons like low latency, more effort should go into onloading perception onto the phone.

Making Every Bit Count in Wide-Area Analytics

Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek Pai, and Michael J. Freedman, Princeton University

Wide-area analytics need to cope with huge data volumes that exceed and outgrow the available bandwidth. Because not all data can be transmitted to a central location for analysis, existing systems make static decisions about what data to collect. They incur high costs for collecting (too) much data, yet are unable to obtain more data retroactively if the need arises.

Ariel Rabkin presented an alternative architecture in which full data is stored close to where it is collected. The data is then aggregated, summarized, and transmitted to the user with a precision and granularity that meets bandwidth constraints. The architecture supports reasoning about the bandwidth requirements of queries. Users can interactively define a policy that controls how results degrade gracefully as bandwidth changes. The OLAP cube is the chosen data model, because it supports merging, summarizing, and aggregating data automatically according to this policy.

When Doug Terry (MSR) asked about other data models that have been considered, Ariel replied that they had looked at SQL tables and MapReduce tuples. SQL tables require too much semantic awareness, especially in the presence of missing data. Alex Snoeren (UCSD) recalled a similar, more general system where custom merge procedures could be specified for every data element. Ariel replied that such merge procedures are difficult to write for rich data, and hard to optimize compared to OLAP cubes. Peter Bailis (UC Berkeley) asked how the system compares to the Tiny Aggregation Service (TAG) used in sensor networks. Ariel explained that the focus is less on reliability and more on using the bottlenecked wide-area link as efficiently as possible.

QuarkOS: Pushing the Operating Limits of Micro-Powered Sensors

Pengyu Zhang, Deepak Ganesan, and Boyan Lu, University of Massachusetts Amherst

Pengyu Zhang presented work that pushes the operating limits of tiny sensors, such as medical implants or self-powered cameras. These harvest energy from temperature gradients, electromagnetic waves, or ambient light to charge energy buffers with a capacity of only few µAh. This severely restricts the amount of work that can be done in a single charge-discharge cycle, and precludes the use of conventional sensor-network operating systems.

QuarkOS fragments tasks as much as possible so that individual fragments stay within the energy limits. QuarkOS efficiently measures available energy and inserts sleep gaps within fragments to recharge the energy buffer. Passive RF communication is given

as an example: fragments consist of transmitting a single bit. Another example is image sensing, where sleeps can be inserted between pixels and even within the different stages of sensing a single pixel.

The first question was about time scales. Pengyu answered that one charge-discharge cycle takes about 100µs, and that one image can be sensed in a few minutes. Somebody then asked how much energy could be saved by this technique. Pengyu replied that QuarkOS does not reduce energy consumption but extends the operating limits of sensors so that they can still execute tasks, albeit slowly, when limited energy is available. Mike Freedman (Princeton) asked at what scale QuarkOS can be applied. Is there a niche between battery-powered devices running conventional sensor OSes and micro-motes running without OS? Pengyu answered that their experiments used the Intel WISP architecture, which fits into this category. These devices have the advantage of being much easier to use than really small motes, where functionality needs to be embedded in hardware. John Ousterhout (Stanford) inquired about the limits of the power buffer. Pengyu explained that larger buffers are possible but disadvantageous: they require over-proportionally longer charge times, need more energy to reach the operating voltage, and cause more heat to be emitted during the discharge.

Open Mike

The open mike session started with Jonas Wagner (EPFL) asking whether partial information from low-rate video processing or low-bandwidth wide-area analytics is really more beneficial than the traditional case where users see full information or none at all. Ariel Rabkin replied that partial information is less scary than it sounds, and definitely useful.

The discussion continued around onloading vs offloading tasks to phones. There are many forms of offloading, some of which are well received. For example, Web sites can be fetched and rendered in the cloud, and be streamed to the phone at the right resolution.

Another topic that was raised was whether hardware could help with fragmenting tasks into even smaller units than what is possible with QuarkOS.

Be More Tolerant, but Not Too Tolerant

Summarized by William Jannen (wjannen@cs.stonybrook.edu)

Failure Recovery: When the Cure Is Worse Than the Disease

Zhenyu Guo, Sean McDirmid, Mao Yang, and Li Zhuang, Microsoft Research Asia; Pu Zhang, Microsoft Research Asia and Peking University; Yingwei Luo, Peking University; Tom Bergan, Microsoft Research and University of Washington; Madan Musuvathi, Zheng Zhang, and Lidong Zhou, Microsoft Research Asia

Zhenyu Guo began with an explanation of Microsoft Azure's leap day bug as an example of how efforts to recover from faults can actually do more harm than help. He analyzed service failures at major companies, and described three of several categories of common misbehaviors: resource contention, "recovered" software bugs, and service dependencies. Zhenyu argued that any failure recovery effort should be engineered to do no harm, because many of the bugs he described led to cascading failures that brought down many healthy system components when trying to recover from a small number of faults.

Zhenyu noted that one element commonly missing in failure recover design is systems thinking—the process of understanding how things interact with a system as a whole. Some decisions may seem correct locally, but are not necessarily globally correct. Systems thinking must be applied in all phases: design, testing, and deployment.

Petros Maniatis asked how easy it is to determine whether an action will do harm or not. Zhenyu explained that it is not easy, and that they have identified challenges in each step of the development cycle. There is no single solution that can solve all problems.

Someone posited the idea that systems thinking might result in a bunch of ground states that the system falls back into rather than cascading failures. In the context of the cloud, ground states might result in the cloud not processing jobs, and therefore not making money. A guiding principle might instead be "don't lose money," rather than "do no harm." Risking cascading failures might be better than running the risk of not making money. Zhenyu agreed that this is a concern, but said that systems thinking is applicable in many situations.

John Ousterhout wondered whether the real problem was that error recovery code never gets debugged; it happens infrequently, but if developers knew it was there, they would fix it.

Toward Common Patterns for Distributed, Concurrent, Fault-Tolerant Code

Ryan Stutsman and John Ousterhout, Stanford University

Ryan Stutsman noted that many current applications scale to support billions of users, and developers write code that is distributed, concurrent, and fault tolerant. When managing thousands of logical threads of execution, the control flow must be adaptive and recover from failures easily, which impacts the way that programs are written. Developers have no control over when faults occur; traditional imperative code doesn't work, and execution history cannot be relied on. Ryan argues that it is only the state of the current system that really matters, and that programs should take steps based solely on state. While working on RAMCloud, they developed rules, tasks, and pools as a pattern for writing fault-tolerant code.

Ryan described rules, which are predicates based on actions. Actions fire in response to whatever conditions happen to be correct at the given moment. He explained that tasks group rules together with the state that they act on. Each task also has a goal, which is an invariant that the task is to achieve or maintain. Pools

group tasks for a subsystem. In this pattern, execution order is determined by state instead of by some predefined ordering, and the execution order can adapt dynamically.

Mike Freedman noted that one way to think about this is that developers are designing systems that represent finite state machines. But it is more general than that, and you don't want to hard code a set of states; writing with this pattern should use actions and triggers. He wondered whether people using this model often write static state machines. Ryan responded that the patterns he's noticed have not had explicit state tags. The conditions apply implicitly. The model is not really about explicit states, but how to reason locally.

Peter Bailis wondered whether Ryan could compare their approach to rule-based languages like Bloom. Ryan was not familiar enough to speak about Bloom, but he thinks about the problem in a similar manner to how model checkers work: the programmer defines conditions and invariants.

John Wilkes observed that in practice, people actually write little state machines, and he thought that the idea of small-scale state machines applied lightly is a powerful idea. Ryan was concerned with the idea of explicit state machines for reasons of scalability. He would like to be able to reason about a system with just a local view of its state.

Escape Capsule: Explicit State Is Robust and Scalable

Shriram Rajagopalan, IBM T. J. Watson Research Center and University of British Columbia; Dan Williams and Hani Jamjoom, IBM T. J. Watson Research Center; Andrew Warfield, University of British Columbia

Shriram Rajagopalan noted that cloud infrastructure scales, and applications should be able to scale easily on that infrastructure as work increases. He proposed the capsule abstraction, a modification of applications and operating systems so that they support scaling at session granularity. The proposal would decouple sessions from applications; mobile sessions would allow balanced scale-out and scale-in, and replicated sessions would allow efficient and transparent fault tolerance.

Each layer must annotate the state that it wants to export, and each capsule must explicitly name its dependencies. A vertical chain of dependencies is called a "slice," which can represent the entire running state of a session. A centralized entity would be responsible for knowledge of capsules at each layer, and it would be able to unplug a slice, move it to another machine, and then plug the capsule back in at the destination. Shriram argued that elasticity and fault tolerance support should be provided at the system level, which the capsule abstraction provides.

Steve Muir commented that capsules were conceptually similar to Google's app engine, and he inquired about the tradeoffs of being intrusive. He noted that for many Web applications, the failure model is simply to drop the connection and restart. Shriram replied that if a single app engine is overloaded, there is no way to shed load dynamically and wait for the request to terminate. App engine scaling occurs at request boundaries.

Erez Zadok inquired as to which entity is responsible for detecting and setting dependencies. Shriram replied that the developer of every layer is responsible for setting dependencies and for registering the capsule. Erez followed up by asking about a case where there are many dependencies and inter-dependencies, to the point that it is cheaper to migrate the whole VM. Shriram noted that most session-based applications do not have dependencies that are so widespread.

Peter Druschel (MPI-SWS) noted that capsules were cheaper than process migration, but more intrusive. Historically, process migration has lost out in favor of VM migration, and Shriram was asked what made him think this trend would reverse. Shriram contended that there is a tradeoff; the coarser the granularity of migration, the less benefit in terms of fault tolerance and elasticity.

Timothy Roscoe asked which sessions would work well in the model. Some sessions might be hard to slice, and for sessions that are short-lived, there would be no point to migrating. Shriram said that for servers with millions of requests per second, this would not make sense, but that normal Web commerce applications have sessions that are not short-lived. A few minutes is more than enough time to overload a machine, and it is a large enough window that a machine can fail, causing a loss of all session state.

Open Mike

The session began with a discussion of Ryan Stutsman's work. Petros Maniatis wondered about the case where two rules created an infinite loop, where each triggered the other. Ryan responded that there is no way to prevent programmers from writing infinite loops, but goal states help. If reaching a goal state takes too long, log messages are generated to help identify the problem. How one could ensure that atomic session code could be kept error free, specifically in the case of memory allocation failure, was also asked. Ryan responded that due to the expense of malloc, they mostly use preallocated buffers. He said that large external failures cannot be ignored, but local error handling can be done. Ariel Rabkin noted that a consequence of state machines being implicit is that it becomes difficult to ensure that progress is being made. Ryan commented that timers help, just as they help to detect infinite loops.

Erez Zadok shifted the discussion back to cascading failures. He noted that many of the examples from Zhenyu's talk suggested that a global view would allow better job handling and recovery. He noted that it might be difficult for a centralized controller to manage large systems, and wondered if a distributed version was considered. Erez likened the situation to current discussions in the world of electrical grid systems, where buildings or city blocks could disconnect themselves from the grid in the case of failure.

Zhenyu replied that restricting failures to containers would help. He also noted that reusing existing failure detection mechanisms is useful.

The session concluded with further discussion of escape capsules and the difficulties that arise when retrofitting capsules to software stacks that were not designed with capsules in mind. Shriram noted that developers may not identify all state that needs to go into a session, and that plugging and unplugging capsules is not an easy job, especially in the presence of unpredictable processes like garbage collection.

Biiiig

Summarized by Seungyeop Han (syhan@cs.washington.edu)

Large-Scale Computation Not at the Cost of Expressiveness

Sangjin Han and Sylvia Ratnasamy, University of California, Berkeley

Sangjin Han presented Celias, a new programming model for large-scale computation. He started by reviewing the MapReduce family (including Dryad and Spark). Although those frameworks support bulk transformation of immutable data, they are not well suited to fine-grained updates on the data set. In their experiments with an iterative MapReduce job for k-hop reachability, they found that overhead takes more than 95% of the whole computation. Further, MapReduce cannot handle dynamic dataflows evolving at runtime. Sangjin proposed a new solution to fix those problems while preserving scalability and the fault tolerance properties of MapReduce.

Their programming model, Celias, is based on the classic programming model, Linda. Whereas Linda uses the process model and does not have any automatic scaling or fault tolerance features, Celias introduces microtasks as the computation model and uses tuplespace as data model. Microtasks are written as signature and code, and are triggered by the availability of tuples that match with the signature. The used input tuple is then automatically replaced by the output tuple. This programming model allows automatic scaling and fault tolerance without the intervention of programmers. Additionally, Sangjin noted that Celias is at least as expressive as MapReduce.

Matt Welsh (Google) commented that sometimes the immutable property is important, especially for rerunning as a batch, and it is important to find killer apps. Michael Freedman (Princeton) said that small tasks would kill performance with frequent I/O. John Ousterhout (Stanford) asked about the consistency issue. Sangjin answered that Celias is relying on atomic operations to ensure that updates are consistent. Petros Maniatis (Intel Labs) asked whether Optimus over Dryad would not solve the problem. Sangjin explained the approach is more like SQL and SQL query optimization and does not give the expressiveness that Celias provides.

When Cycles Are Cheap, Some Tables Can Be Huge

Bin Fan, Dong Zhou, and Hyeontaek Lim, Carnegie Mellon University; Michael Kaminsky, Intel Labs; David G. Andersen, Carnegie Mellon University

Bin Fan presented a new hash table that can serve a very large number of entries entirely from memory. Their target is when keys could be large whereas each value costs a few bits. He showed an example of the hash table storing UserID → online/offline. In the traditional hash tables storing those entries, some rows are not utilized. Additionally, storing keys to avoid collision takes another large space. Overall, it requires O(k+v) bits/entry.

By contrast, Bin's team suggested a new data structure to save memory. The core idea is to throw away the keys and to do brute force to avoid collisions. To do so, their algorithm, SetSeparation, enumerates hash functions in a hash function family to find the hash function that maps all keys in a group to correct values. Then, it records the parameter to get the hash function. Dividing the entire input into small groups, their scheme can handle a large number of keys/values. By the algorithm, their data structure uses only 0.5 + 1.5v bits/entry. Bin noted that the algorithm has a caveat that it cannot handle a membership function because it does not maintain keys by itself. In evaluation, SetSeparation uses only 3.88 MB for 16 million entries, whereas the STL (Standard Template Library) map uses 869.46 MB and the lookup speed is faster.

Jonas Wagner (EPFL) asked how Set Separation handles updates. Bin answered that it needs to keep track of which keys are in the group in external storage. Volodymyr Kuznetsov (EPFL) commented that STL map is not a hash table and asked whether lookup and update cost would depend on key-size. Bin noted they were using a hash map and lookup is still constant although a little bit tricky. Michael Freedman (Princeton) asked whether figuring out which group the query key is in is not key-dependent. Bin replied that determining it is again based on hashing. Roxana Geambasu (Columbia) asked about concrete applications, noting that it has restrictions. Bin mentioned software routers as one example. Dan Williams (IBM Research) commented that it would be expensive for the cases with longer values. Bin said that it needs to be done per-bit for a multi-bit case, and the benefit decreases for longer values.

Wanted: Systems Abstractions for SDN

Sapan Bhatia, Andy Bavier, and Larry Peterson, Princeton University

Sapan Bhatia started by noting that iptables were functioning as a Swiss Army knife for many network configurations: while iptables is a powerful tool, it has the reputation for being tedious to use and error-prone. Additionally, changing configuration leads to resetting state, such as policies or routing entries. The research community has provided useful results, including new network architectures, domain-specific languages (such as Click), OS extensions, and finally SDN. In practice, however, nothing is changed and configuration still involves iptables.

Sapan explained that they have taken the best of academic ideas with standard tools. He presented NativeClick, which combines Click Modular Router's language to specify the graph and native runtime overlaid on the Linux networking stack. More specifically, elements and ports of Click are replaced with executable scripts and virtual links. Key mechanisms allowing this are from the network container to isolate route tables, policies, and virtual links. For connection to SDN, Sapan noted that expanding SDN to the end host is important. Also, he showed an SDN perspective consisting of vdev, controller, and processes in a middlebox.

Andrew Baumann (MSR) asked how to debug iptables since it requires understanding the Click abstraction. Sapan noted that it is an open problem in the generated codes, and current iptables itself is hard enough to debug. John Wilkes (Google) asked about evaluation. Sapan said that it is more community-driven, and users do not complain about it. Shriram Rajagopalan (UBC) commented that the SDN connection is a bit weak. Sapan noted it is about how you do middlebox functions and that the systems and the SDN approaches meet, since it achieves the end-result of SDN through OS functions.

Open Mike

The open mike session started with a question from Siddhartha Sen (Princeton) to Bin Fan about whether inserting lots of new keys could affect the performance. Bin answered that each group can handle a small number of keys, and thus more than 30 keys per group may require rehashing. Erez Zadok (Stony Brook University) continued with a comment that this is somewhat similar to Bloom filters and worth exploring the similarity. Bin replied that the difference is that their mechanism does not make any mistakes for the known keys, which a Bloom filter might do. One person from MSR wondered whether Bin's team used the same code for underlying hash functions in CHD (the Compress, Hash, Displace algorithm) when they evaluated. Bin answered they used the reference code from Google; a coauthor, Hyeontaek Lim (CMU), added that a number of entries would degrade CHD performance as well, and thus changing the underlying hash function would not change the trends.

There was a big discussion about applications for system research. Brian Noble (U Michigan) said that everyone should spend time finding someone doing computationally intensive projects. Timothy Roscoe (ETH) mentioned that computational finance and sociology will be interesting fields in terms of applications, and John Wilkes (Google) added biology and medicine. Petros Maniatis (Intel Labs) said that applications do not need to be solid ones, but it does make the work plausible. John Wilkes commented that for something big, we do not have an application yet, and we need to think not of applications, but problems and how we can solve them. Matt Welsh (Google) said that we have to get inspiration from problems out there and need to do generalization. Timothy Roscoe said that he had found someone with a big problem: he had teamed up with people who had fled the big banks and investment companies, as well as people still working at Credit Swiss, to do work on financial modeling. He has also worked with the Swiss Federal police in tracking counterfeited watches shipped around the world.

Someone commented that many people need help identifying their problems. Brad Karp (UCL) gave an example of block boundaries that are used for many other problems, although not for applications, but it is a fundamental problem of bigger systems.

Catching Up in the Clouds

Summarized by Deian Stefan (usenix@deian.net) and Edward Yang (ezyang@cs.stanford.edu)

The Case for Tiny Tasks in Compute Clusters

Kay Ousterhout, Aurojit Panda, Joshua Rosen, Shivaram Venkataraman, Reynold Xin, and Sylvia Ratnasamy, University of California, Berkeley; Scott Shenker, University of California, Berkeley, and International Computer Science Institute; Ion Stoica, University of California, Berkeley

In data-parallel computing, the straggler problem arises when a single task runs at a much slower rate (e.g., because it's running on a slow machine) than other tasks, slowing down the whole job. Yet, we typically schedule large batch tasks to ensure high cluster utilization. This not only amplifies the straggler problem, but also gives rise to another problem: cluster responsiveness. By running long batch tasks, short interactive jobs may need to wait on the order of seconds or minutes before being serviced, effectively rendering the cluster unresponsive.

To address these issues, Kay Ousterhout argued that all data-parallel jobs should be broken down into tiny tasks. This addresses the straggler problem by ensuring that workloads are evenly distributed across machines; fine-grained scheduling ensures that slow machines are assigned fewer tasks than fast machines. A simulation on Facebook workloads showed that using tiny tasks would improve the response time by roughly 5x. In a similar fashion, the tiny tasks paradigm bridges the gap between cluster utilization and responsiveness: long-running batch jobs are broken down into thousands of tiny tasks, allowing short interactive jobs to be interleaved as launched.

There are many challenges in implementing an architecture that employs the tiny task paradigm. To narrow the challenges, the authors focus on applying the model to data-parallel computations similar to MapReduce. In such a scenario, a task is typically I/O bound (reading input data stored on disk), and, to ensure high disk utilization, a tiny task must run for at least a few hundred milliseconds—a duration they argue that is acceptable even for Web applications. This is challenging as it requires changing the programming model to break a job into many tiny tasks, reducing the launch of a task to a few milliseconds, implementing a task scheduler that handles millions of decisions per second, and changing the underlying distributed file system to handle many small reads; however, using similar techniques to Spark and FDS, the authors

believe they can address some of the concerns; developing a practical architecture, although promising, is part of their ongoing work.

Mike Schroeder (MSR) asked for a characterization of the jobs for which the straggler problem was not solved by their solution. Ousterhout noted that tiny tasks require a change in the programming model, but programmers can ignore this and, for example, can still write code that contains infinite loops—in such cases, tiny tasks won't do much to improve the situation. Jeff Mogul asked how long a job should be, as opposed to how long it can be (i.e., short enough to read 8 MB as to use the disk efficiently). Ousterhout noted that the few hundred milliseconds is consistent with the shortest duration of data-analytics jobs they've observed in practice. Hyeontaek Lim pointed out that dividing a 40,000-task job into 4 million won't necessarily be "better"; what size jobs should be sub-divided? Ousterhout explained that they had looked into the space to find characteristics of different jobs and found that jobs with a few tasks were the ones with long-running tasks; finding the precise point where diving into more tasks becomes inefficient is part of future investigation.

Using Dark Fiber to Displace Diesel Generators

Aman Kansal, Microsoft Research; Bhuvan Urgaonkar, Pennsylvania State University; Sriram Govindan, Microsoft

High availability is a lot of work. A server may be protected against power failure by a UPS; but this is no good if your network gateway goes down: datacenters must also install diesel generators to protect against utility failure; but this, too, fails in the event of physical disaster, so your data must be georeplicated. Highly available services are deployed with multiple layers of redundancy, and this redundancy is expensive. Because high availability services must always be georeplicated, Aman Kansal suggested relying solely on georeplication for availability, reducing the availability needs for any given datacenter.The authors argue that "Geo-distributed Bunches of Datacenters" (or GBoDs) could be practical, but there are a number of questions to answer. For one, how much can one reduce DC availability before global availability is affected? Assuming independent failure, one can calculate this out: for n=10, one can do with 0.1% failure probability rather than 0.001%. A bigger question is how applications need to adapt to this new scheme. Some methods of georeplication, such as sharding distributed state, no longer work as everything must be replicated everywhere—addressing this is an open research problem. Bandwidth, however, is not a problem: the authors propose that the dark fiber connecting these datacenters be used to carry out the large amounts of data transfer necessary to perform full replication.

Timothy Roscoe pointed out that building a new datacenter takes a really long time: on the order of seven months, which is quite different from spinning up a new server. Jeff Mogul noted that as the reliability of single datacenters decreases, the error bars on your availability calculation increase. One might do OK if there is an error margin built into your availability figures; but that margin costs money, exactly what GBoDs are trying to save. Edouard Bugnion asked which workloads could be distributed this way, and Aman answered that without software redesign, read-only software is the only thing that can be done; applications with real-time data writes are considerably more difficult.

Towards Elastic Operating Systems

Amit Gupta, Ehab Ababneh, Richard Han, and Eric Keller, University of Colorado, Boulder

Amit Gupta said that one of the main benefits of cloud-based systems is the ability to elastically change the amount of resources allocated to an application according to demand; however, we presently place the burden of elasticity on apps: an app has to, a priori, be designed to operate in a cloud environment. The developer must design the app such that it can distribute the workload, on demand, among different instances; handle data consistency issues (e.g., sharing across instances); and monitor load as to decide when to expand or contract the number of nodes.

Rather than continue building apps with elasticity in mind, Gupta argued for making elasticity an OS primitive. ElasticOS would allow applications to be built without any notion of elasticity, while transparently expanding and contracting to accommodate different workloads. To this end, they propose using elastic page tables, i.e., page tables that map virtual addresses to machine/physical addresses, as a way to allow an application to expand when memory on other nodes becomes available and is in demand. Different from previous distributed shared memory (DSM) systems, they, however, do not replicate data pages across machines. Instead, paging-in remote tables results in them being moved from the remote machine. This avoids the need for coherency protocols that have plagued DSM systems; however, to take advantage of locality, they propose migrating the process/thread execution context once the number of pages that are being pulled in reaches a certain threshold. Unlike data pages, this can be quite efficient because caching multiple copies of code pages does not require DSM-like protocols. Gupta concluded the talk with the remark that although various issues (e.g., fault tolerance and elastic network I/O) need to be addressed, their preliminary Linux implementation has shown promising measurements.

Jay Lorch (MSR) was skeptical about the approach, as it wound up leading researchers on the same path as DSM. In response, Gupta noted that their work differs from the DSM efforts in two important ways: DSM heavily relied on replication and kept execution context fixed (except for process migration); in their work, they keep a unique copy of data and move execution contexts when appropriate. Andrew Warfield noted that moving contexts around is expensive (because it requires transferring roughly a page of context information) and asked why moving the context to the data is a good idea (because this happens often when stretching to a large number of nodes). Gupta noted that they adopted a hybrid approach: they pull data until they notice that they can

exploit locality, and at that point they jump. He further noted that for certain workloads this approach may not work, but this requires further investigation. Timothy Roscoe brought up the issue of memory efficiency: if code pages are replicated to allow fast context transfers, at what point does this approach become inefficient? Gupta noted that in data-intensive applications, such as MySQL, the number of code pages is much lower than the corresponding number of data pages, so they do not anticipate a large overhead if code is carefully replicated across a (part of the) datacenter. The last questioner asked whether there is any reason to believe that cluster-wide parallelization is going to be better than multicore. In response, Gupta noted that a process on a single node is inherently bound by memory and they intend to break that barrier.

Open Mike

Rik Farrow provided the quote of the session: "I think you live in an alternate reality called Google."

Everyone seemed to agree about tiny tasks for cluster computing (except one guy from Berkeley), so the conversation turned to a discussion about GBoDs and elastic computing.

The subject of datacenters was close to the heart of many of the industrial members of the audience. Two interesting topics came up during the ensuing discussion. The first was political reasons why applications may not be georeplicated; for example, a country may have strict data privacy laws that prevent data from being replicated across its borders. Jeff Mogul mentioned that this was exactly the case, and that they had implemented selective georeplication. John Wilkes (Google) brought up the cost calculation that companies are constantly doing when considering datacenter administration. Some infrastructure has 11 datacenters deployed to serve 10 datacenters' worth of load, with the last datacenter running compute jobs on the extra capacity. As opposed to infrastructure such as Google AppEngine, which has excessive redundancy, GBoDs may not be a win in such situations. Additionally, when a datacenter goes down, there is the cost of all the hardware that is not being utilized in that datacenter; one participant noted that making sure that this hardware is not wasted is worth at least some money.

The response to the elastic computing talk had been considerably more prickly, and so Jeff launched a new discussion by pointing out that ElasticOS was targeted at being fully backwards-compatible, whereas tiny tasks and datacenters asked programmers to change their programming model. "Aren't we underestimating the value of not changing applications?" Matt Welsh responded that at Google, "We are constantly changing our applications to adopt new programming models." This led to Rik Farrow's response: "I think you live in an alternate reality called Google." There was some debate whether or not MapReduce was an example of a new programming model that had been rapidly taken up by non-Google programmers. Lim countered by stating that Hive/Pig were used by people who looked at MapReduce and said, "We want SQL." Depending on who you ask, the majority of MapReduce jobs are written in these languages.

Others were confused about whether or not ElasticOS bought anything in an era where machines with 1 TB memories could be purchased. Moving around all this data, especially in a failure tolerant way, would be difficult. "At some point," one participant commented, "won't brute force just win out?" The authors acknowledged this, and argued that you'd have to make locality assumptions about the usage of 1 TB of memory.

Correct, Secure, and Verifiable

Summarized by William Jannen (wjannen@cs.stonybrook.edu)

Toward Principled Browser Security

Edward Yang, Deian Stefan, John Mitchell, and David Mazières, Stanford University; Petr Marchenko and Brad Karp, University College London

Deian Stefan noted that the Web has evolved into an application platform. And although traditional operating systems provide applications with page protection and file system permissions, the browser must rely on the same origin policy (SOP) to protect data. There are exceptions to strict isolation in the SOP; on the one hand, these exceptions allow developers to build complex, information-sharing apps; on the other hand, exceptions can lead to leaks of sensitive data.

Deian listed several remedies for SOP shortcomings, such as the content security policy (CSP) and cross-origin resource sharing (CORS), but noted that such measures are coarse-grained, static, and inflexible. He proposed a more principled approach—to use information flow control (IFC) as a browser security primitive. Browser-based IFC would do more than just emulate the SOP; it would allow execution of untrusted code on sensitive data. A strict base policy could enforce origin non-interference, but the framework would allow flexibility and fault isolation.

Matt Welsh asked about the proposal's implications on both browser and Web API designs, and whether it would require a change to all browsers and all API code. Deian noted that the proposal would require browser modifications, but it would not require a modification of JavaScript; it would be just another API that developers could use. Deian was then asked about memory and performance overheads, and the potential implications that overheads would have in the browser performance war. He replied that although he did not have numbers on hand, there would be no impact on the performance of existing code. The proposal is effectively an opt-in and coarse-grained approach. Don Porter requested some implementation insights. Deian responded that it is implemented as a whole new API. They leverage Gecko's compartment model, with all implementation done at the language level.

Deian was asked to discuss the differences between their proposal and FlowFox from CCS. He explained that the FlowFox mechanism was for JavaScript only, was not opt-in, and could break existing Web sites; also, it does not support declassification. Ashvin Goel (U Toronto) asked how to ensure that attackers could not simply bypass checks, especially in the presence of browser bugs. Deian noted that avoiding bugs is difficult, but that they leverage Gecko's compartment model to isolate memory spaces.

Volodymyr Kuznetsov (EPFL) asked about side channels. Deian commented that this is an extension of their previous work that does address some side channels, but with respect to external timing channels there is not much they can do. Peter Bailis asked whether an opt-in policy would allow adversaries to hide in legacy content. Deian clarified that the proposal would not impose on existing Web sites, but a Web site that uses the API would be protected.

-OVERIFY: Optimizing Programs for Fast Verification

Jonas Wagner, Volodymyr Kuznetsov, and George Candea, École Polytechnique Fédérale de Lausanne (EPFL)

Jonas Wagner noted that there are many tools that prove the safety and correctness of software, but that these tools are rarely used in practice because often they are slow or hard to use. One reason that existing tools are slow is because they receive the wrong kind of input—a performance-optimized binary; the time it takes to verify a program can be made significantly faster by compiling specifically for verification instead of for execution on a CPU. As an example, branches are costly for verification, and equivalent branch-free code often can be verified more easily.

Jonas proposed a compiler switch to enable verification optimizations, much in the way -g is used for debugging, and -O3 for performance. The -OVERIFY flag would signal the compiler to preserve high-level information, favor optimizations that ease verification, annotate the program, and generate runtime checks so that verification tools can easily detect bugs. They actually have an implementation that they have tested.

Ariel Rabkin asked whether performing these optimizations inside the compiler or inside the verification tool itself makes more sense. He also wondered whether verification time was really a limiting factor. Jonas noted that the time it takes to verify is important; a drastic cost reduction would not only save developer time, it could change the ways that verification tools were used, to the extent that they potentially could be used at every commit. And one of the principal advantages of using a compiler flag is that it does not require any changes to existing verification tools.

Ariel then asked if the same tweaks are valuable for all verification tools. Jonas explained that there are different types of tools; their prototype, -OSYMBEX, generates code optimized for symbolic execution tools. Martín Abadi then posed an idea: what if a compiler could generate several different versions of the binary, each optimized for verifying a particular property? Jonas noted that this would work particularly well for finding concurrency bugs.

When Andrew Birrell (MSR) asked about high-level information that can't be transferred down to assembly, Jonas remarked that a binary with debugging information has complete source code, but that not all information is necessary. High-level types, and information about which variables are local, global, or thread local would be helpful.

Global Authentication in an Untrustworthy World

Martín Abadi, Andrew Birrell, Ilya Mironov, Ted Wobber, and Yinglian Xie, Microsoft Research

Andrew Birrell gave a quick recap of authentication with X.509 certificates, noting many positive features: authentication is completely decentralized, non-hierarchical, and worldwide. Additionally, X.509 is pervasive and quite secure; however, Andrew pointed out that being quite secure is almost as bad as not being secure at all. He used a few high-profile examples of failures to prove this point. The underlying problem is the large scale of trust—the relying party trusts every CA in the delegation chain, not just the root or the leaf. Intermediate CAs are all uniformly powerful and can write a certificate for any name. Andrew argued that although non-hierarchic authentication is essential, uniform trust of worldwide CAs does not work. Local policies are a better approach.

Andrew then discussed the details of their data set. A 2010 EFF data set was parsed and then supplemented with additional data collected in 2012. In total, 7.8 million certificates were acquired from 22.7 million TLS handshakes, and the details were organized in an SQL database. Although the database enables ad hoc queries, the data is too large for ad hoc analysis; they instead performed cluster analysis, choosing a set of 18 features that were thought to be interesting, including key length, country, trusted root, etc. The result was a set of 28 tight clusters with few outliers.

Andrew presented uses for the data set, such as a user-controlled policy engine. The database could be queried to make trust decisions. Policies could be designed by experts and selected by the end user.

Mike Freedman wondered why SPKI never took off, given that it allows chained delegation. Andrew responded that SPKI allowed Web-of-trust-like things, but clearly there was not enough demand. People seem quite happy with the current situation using X.509, except that it breaks two times per year. Deian Stefan asked about data access. Andrew hoped that Microsoft would allow the data set to be made public, but he noted that the 2010 EFF data set is available.

Petros Maniatis asked about the implementation of any policies that might have made sense for Microsoft, and whether Andrew had evaluated how many Web sites had such policies "turned off." Andrew joked that had they done this evaluation, it would have been an SOSP paper, but they are currently working on it.

Automated Debugging for Arbitrarily Long Executions

Cristian Zamfir, Baris Kasikci, Johannes Kinder, Edouard Bugnion, and George Candea, École Polytechnique Fédérale de Lausanne (EPFL)

Cristian Zamfir explained that the debugging process, identifying and fixing the root cause of a program failure, differs during development and production. During development, the gdb record option can be used to reverse step from the point of failure, but in the production world, a core dump from a segmentation fault cannot be reverse-stepped. Although production level record support is possible, overheads may be prohibitive. The question, then, is what can be done with limited information in production systems?

Cristian proposed reverse execution synthesis (RES), which takes as input a program and its core dump, and outputs an execution suffix that would lead to that core dump. He noted a key insight is that there exists a large class of programs for which the root cause is close to the actual point of failure, making the search space manageable; however, the challenge is inferring the paths. This can be done by recording constraints through branches and checking against the core dump state. By applying this process recursively, the system can build an incrementally larger execution suffix. As long as the start of the path contains feasible values, the execution suffix is guaranteed to reach the error state. RES can debug arbitrarily long programs with no runtime overhead.

Steve Hand wondered how many distinct paths were often observed. Cristian replied that RES works well for small concurrent programs, and that they are able to synthesize unique suffixes in about a minute. But, in general, a program that overwrites much of its state would result in many execution suffixes.

John Wilkes asked whether logs could be leveraged. Cristian said that logs could provide path information, which is important. They would not provide full paths, but they would provide specific points, which could disambiguate state.

Petros Maniatis asked about the tradeoffs of checkpointing at runtime, and then combining forward and backward search. Cristian replied that fast checkpointing might be something worth using and could potentially be used to validate the feasibility of states. But his position is to do as much as possible without recording; checkpointing is a form of recording.

Jeff Mogul asked whether the compiler could be leveraged, like -OVERIFY, to generate log entries at specific points where reverse stepping would be difficult. Cristian said that the compiler could try to use less overwriting, and that they are trying to use copy-on-write when possible.

When John Wilkes asked for project insights, Cristian replied that the project is still in its beginnings. Execution suffixes are currently on the order of hundreds of instructions, but it depends on the specific program and how much rewriting it does. He noted that without debugging symbols, a control flow graph is necessary in order to determine possible paths.

Open Mike

George Candea wanted to know how comfortable people were with putting specialized code in programs solely for post-mortem analysis. He was curious about the range of measures with which people were comfortable. Matt Welsh wanted clarification as to whether George was asking about developers, libraries, or runtimes. George responded that that was the point of his question. He thought some people might be uncomfortable with a 5% overhead, but Matt thought that 5% was absolutely fine because the information gained was invaluable. John Wilkes noted that monitoring systems generate several percent overhead, so overheads under one percent are well within the acceptable threshold. Jeff Mogul said that what is unacceptable is logging information that causes privacy concerns.

Matt Welsh asserted that reviewers should make sure to avoid punishing papers when the overheads are over these thresholds. Also pointed out is that just because a technique is not acceptable for production, it is still worth reading. Mike Freedman commented that it is also important for authors to be careful about how they calculate overheads. Erez Zadok reiterated that acceptable costs are dependent on the application. NASA's Jet Propulsion Labs might be willing to accept overheads of 20–30% for a Mars rover, so the community shouldn't set simple thresholds. John Wilkes added that thinking about the cost to fix bugs is also important. There should be more flexibility than just one magic number. What we would like is a range of things and different choices. Overheads accumulate, so thinking about priorities and making sure that important features are the ones that are ultimately incorporated is important; what might be acceptable on a server might not be acceptable on a phone.

Petros Maniatis asked about the role of hardware. He noted that Intel provides branch information such as last branch records (LBR), but that in terms of performance, these things are not free. Intel must prioritize things, too, so if the software community would come to a consensus, then hardware designers could make these decisions.

A general comment was that the session's debugging papers assumed a C-code environment, but there also is interest in managed language runtimes. A lot of production code is written in languages such as Java and C#, and this might be an easy place to add diagnostics. An open question was how general can these tools be made.

Something Old, Something New, Something Hot

Summarized by Cristian Zamfir (cristian.zamfir@epfl.ch)

Operating System Support for Augmented Reality Applications

Loris D'Antoni, University of Pennsylvania; Alan Dunn and Suman Jana, University of Texas at Austin; Tadayoshi Kohno, University of Washington; Benjamin Livshits, David Molnar, Alexander Moshchuk, and Eyal Ofek, Microsoft Research; Franziska Roesner, University of Washington; Scott Saponas, Margus Veanes, and Helen J. Wang, Microsoft Research

David Molnar explained that augmented reality (AR) applications impose new challenges on operating systems for several reasons. First, AR applications must deal with potentially sensitive data that gets mixed with user input, which calls for a more fine-grained permission system. David showed how the raw video input stream may contain user faces and private information, yet any application can access this information, so this will not work with AR applications that multiplex access to the same video stream. Second, the window system will have to be updated in order to handle 3D objects from multiple applications, as opposed to the square windows we have today. Third, AR systems have to deal with continuous inputs (e.g., gestures) that are also inherently noisy (e.g., an object may be confused with an arm).

David pointed out that given the emergence of such systems, these challenges (especially the privacy-related ones) will have to be solved before the legislation is updated in probably 2–3 years. Otherwise, without some privacy guarantees, AR systems may even be officially banned from certain contexts.

Michael Freedman (Princeton) asked what lessons from Web mash-ups can be applied in this area. David mentioned that the work on clickjacking defense can be used. Another issue is the Same Origin Policy, which does not yet exist in AR systems, but there is room to innovate in this area.

Steve Muir (VMware) asked if the OS should manage the access to private data. David argued positively, and briefly described his upcoming paper in USENIX Security on how to provide visual explanations to users of what the requested permissions allow applications to access. Stefan Bucur (EPFL) asked whether information flow control could help. David agreed that is a good direction for exploration. Peter Druschel (MPI) asked whether there will be a "one size fits all" set of abstractions for the AR applications. David said that the answer is likely yes, since this model will be easier to use by developers.

Solving the Straggler Problem with Bounded Staleness

James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Gregory R. Ganger, and Garth Gibson, Carnegie Mellon University; Kimberly Keeton, HP Labs; Eric Xing, Carnegie Mellon University

James Cipar introduced Stale Synchronous Parallelism, a model that maps to scientific applications and can tolerate stragglers. The key idea is that this model allows applications to tolerate significant delays in some threads. Preliminary results with an early prototype show that increased staleness can mask the effects of occasional delays. The model also detects when data becomes too unsynchronized, and synchronizes threads to avoid unbounded staleness. An important open question for ongoing work is how to automatically tune the requirements of the application regarding freshness.

Doug Terry (MSR) asked whether the staleness bound impacts convergence and James answered that, in their experience, it is important. Mike Schroeder (MSR) asked whether their method works with non-transient delays. James answered that their approach supports temporary delays, like a GC pause or some additional computation done by a specific thread, but it cannot do anything against non-transient delays. Roxana Geambasu (Columbia) asked what other kind of applications this model accommodates. James said they have experience with scientific computing applications, page rank, and machine-learning algorithms that resemble gradient descent. Jonas Wagner (EPFL) asked why performance improves when there are no delays. James answered the staleness model masks some delays. David Ackley (UNM) pointed the authors to related work that uses a similar technique to tolerate transient errors. This technique works for errors, but might apply also to delayed computation.

Lightweight Snapshots and System-Level Backtracking

Edouard Bugnion, Vitaly Chipounov, and George Candea, Ecole Polytechnique Fédérale de Lausanne (EPFL)

Edouard Bugnion introduced the concept of lightweight snapshots, a new state abstraction that provides immutable snapshots integrated into the virtual memory subsystem. Based on the lightweight snapshots abstraction, he proposed a design for an operating system that provides system-level backtracking for arbitrary applications. The design of the backtracking OS leverages modern x86 hardware-virtualization support to perform efficient backtracking and supports configurable scheduling policies.

Edouard gave several examples of applications that can benefit from the backtracking OS (e.g., S2E, a demanding application that implements full-system symbolic execution, and Z3, an SMT solver). He also exemplified the system-level backtracking API using the canonical n-queens example. Their early prototype can already provide backtracking capabilities to complex applications such as Z3, with minimal changes to the application.

David Molnar (MSR) asked whether developers can pass the scheduling heuristic to the OS. Edouard answered this is indeed possible. Andrew Bauman (MSR) asked whether it would be better to move the scheduler outside the OS. Edouard answered that the scheduling policy and the scheduler are decoupled: the scheduler can be in the OS, and the scheduling policy can be set by the application. Edward Yang (Stanford) asked whether the proposed abstraction can be thought of as a faster fork(). Edouard answered that it is more than that, since it is hard to just use fork() and combine it with various search heuristics. Brad Karp (UCL) asked whether privilege separation (as in Wedge, a system built at UCL)

is another application of the proposed system. Privilege separation requires strong isolation, but can this be added? Edouard answered that Wedge was eventually built into Dune. The big takeaway is that one can now envision building domain-specific operating systems.

HAT, Not CAP: Towards Highly Available Transactions

Peter Bailis, University of California, Berkeley; Alan Fekete, University of Sydney; Ali Ghodsi, University of California, Berkeley and KTH/Royal Institute of Technology; Joseph M. Hellerstein and Ion Stoica, University of California, Berkeley

Peter Bailis proposed highly available transactions (HATs) that are available in the presence of network partitions. The CAP theorem shows that it is impossible to provide linearizability in the presence of arbitrary network partitions, and does not directly apply to database transactions. Peter pointed out that even single-node databases do not provide serializability by default, because it is expensive. Instead, they provide weaker consistency models, and many applications work well with these models and can tolerate the arising anomalies to gain performance. However, it is not clear which models can be achieved with high availability.

Their work is about exploring the class of high availability low-latency transactions that can be achieved in the presence of network partitions. Peter proposed techniques based on read or write buffering to provide some guarantees (read committed and repeatable read isolation) for a HAT system, and also described some additional guarantees that they proved are not achievable (e.g., regency bounds and some integrity guarantees).

Brad Karp (UCL) noted that previous papers about Spanner and Eiger mentioned similar social networking examples (e.g., the order of the posts). Brad asked what HAT can provide compared to this other work. Peter answered that there are many existing applications that work with the weak consistency offered by today's databases, so this is a useful programming model. Moreover, the anomalies that would appear under these models do not appear for some applications. For instance, TPCC isn't subject to anomalies from weak consistency, which is why Oracle is TPCC-compliant and offers a weak consistency model. Doug Terry (MSR) argued that one way to implement repeatable reads is to just not allow any transactions to commit when you have a partition. Peter said that with transactions you can have success and abort, so one can abort everything and obtain the liveness property. Their paper contains details on how they define transaction availability. Michael Freedman (Princeton) asked whether the write buffering technique is two-phase commit. Peter answered no and explained the differences.

Open Mike

Byung-Gon Chun (Microsoft) asked how the bounded staleness model compares to the asynchronous lazy synchronization model used in GraphLab. James answered that GraphLab makes assumptions about data locality and would also require modifications to their algorithms to accommodate staleness. Petros Maniatis (Intel) asked whether their work is about figuring out how much staleness can be supported by the applications. James answered that they established a profile of the applications that work, and identified several applications that fit the profile. Steve Hand (Cambridge) suggested that if one speculates, then one may also need to roll back, so they could use lightweight snapshots proposed in the talk by Edouard Bugnion.

Jacob Lorch (MSR) asked how to evaluate which of the consistency models discussed in the HAT not CAP talk is reasonable and can be understood by users. Peter Bailis (Berkeley) argued that it is still an open question what consistency models to run on and not violate the application's integrity constraints. Peter argued this is a great direction that should see more work and exemplified with work from Marc Shapiro at INRIA on conflict-free replicated data types. Siddhartha Sen (Princeton) proposed comparing the code that one would have to write to deal with weaker vs stronger consistency. Ali Ghodsi (Berkeley) commented that Doug Terry's session consistency model already prevents several anomalies that users see, so the big open question is what is the consistency model that is both efficient and prevents most of these anomalies.

Hardware to the Rescue

Summarized by Cristian Zamfir (cristian.zamfir@epfl.ch)

The von Neumann Architecture Is Due for Retirement

Aleksander Budzynowski and Gernot Heiser, NICTA and University of New South Wales

Gernot Heiser's talk was motivated by the plateau reached by CPU frequency and the multicore trend; he proposed a self-modifying data flow graph computation model to replace the von Neumann model. Their model essentially does away with global memory, thus aiming at making it possible to express and implement general purpose parallel computations easier and more efficiently.

A typical data flow computing model is static, and there is no way to express dynamic algorithms and data structures. To address this challenge, they propose a data flow graph that can change itself, change references to other nodes in their immediate neighborhood, create new nodes, etc. They have a partial implementation that takes Haskell code as input and translates it into data-flow assembly.

Ariel Rabkin (Princeton) wondered how synchronization is implemented and asked to see how the proposed design works for something simple like matrix multiplication. Gernot answered that synchronization is entirely done by data flow. He also mentioned that the example he described in the talk is more complex than a matrix multiplication and would work for dynamic data structures. Mike Schroeder (MSR) asked about the next step; where do they plan to get the hardware to implement this? Gernot said they can try to simulate this architecture in software without the performance benefits. Moreover, their work is inspired by a startup that aims to build fully asynchronous hardware.

David Ackley (UNM) said that the answer to all the open questions raised by the talk is coming up with a spatial layout of the graph, which has to be embedded in the hardware, which has to be spatially extended, yet still be finite. Gernot answered that there is commonality between their hardware and the hardware proposed by David at the previous HotOS. They are trying to get away from the global address space yet retain as much of the CS abstractions as possible, thus making the model more easy to program than David's model.

Brad Karp (UCL) asked whether before proposing such a change at the hardware level, one does not have to refute the arguments made by people working on taking a sequential programming model and making it work for multicores. Gernot argued that everyone is trying to tweak the von Neumann model, but these approaches will run out of steam after some scale. He argued that his system has some nice properties that are worth exploring.

Arrakis: A Case for the End of the Empire

Simon Peter and Thomas Anderson, University of Washington

Simon Peter argued that recent hardware devices enable building kernels that allow applications to talk to hardware directly, without OS mediation; the kernel only provides control plane services (e.g., deals with resource reallocation), but applications use a library linked in their address space to talk to hardware directly. One enabler for this design is the fact that hardware is increasingly virtualized. Moreover, I/O devices become faster while CPUs are bottlenecked by frequency, so unmediated access to hardware devices is an important performance-related requirement.

One of Arrakis' several goals is to allow applications to customize OS functionality (e.g., provide protection domains using hardware protection). Moreover, Arrakis is designed to provide device driver safety, by running device driver replicas and ensuring that when one replica crashes, the system does not crash. One important challenge is dealing with the fact that hardware may not provide sufficient virtualization capabilities for meeting all the proposed design goals.

Jeff Mogul (Google) said Arrakis looks like it is partially reinventing the InfiniBand model (which has had this separation for a decade). Simon answered they are trying to generalize that model to other hardware. Steve Muir (VMware) argued that Arrakis needs to support migration and checkpointing to be useful for real-world use cases and Peter agreed. Edouard Bugnion (EPFL) asked what can be learned from the way people build the control/data plane separation in network hardware. Simon answered this was part of their inspiration and that they are already looking at that literature.

Rethinking Network Stack Design with Memory Snapshots

Michael Chan, Heiner Litz, and David R. Cheriton, Stanford University

Michael Chan proposed a redesign of the network stack, which leverages HICAMP (ASPLOS '12), a hardware memory system that supports snapshot isolation. The system allows zero-copy, reduces memory allocations, and works with the existing socket API. The main motivation for this work is that the networking stack uses many memory allocations and accesses, while network I/O speeds are going up. Unlike existing approaches, users do not have to use specific data structures to do zero-copy; instead they can use the application data. Compatibility with the POSIX API is done by simply passing another flag to the malloc() call to use HICAMP memory.

Michael showed how to do zero-copy I/O and how to simplify the DMA process and the NIC design. He also discussed the space and time tradeoff of the design. He ended the talk by arguing that software-hardware co-design can improve OS architecture and solicited ideas for applications to other areas of system design.

Siddhartha Sen (Princeton) pointed out that persistent data structures (some developed by Targent) can be used to efficiently keep multiple copies of a data structure and be able to update it partially. Jacob Lorch (MSR) asked when the hardware will be available. Michael mentioned they have a simulator and plan to make it available to others soon.

Rik Farrow (USENIX) mentioned that their system ends up doing pointer chasing, which imposes some overhead. Michael said there are two additional reads/write when writing duplicate data. Michael mentioned some back-of-the-envelope calculations for network I/O that seem very optimistic (several hundred Gbps), so even achieving 50% of that would be impressive.

Edouard Bugnion asked about the downside when integrating with the cache hierarchy. Michael answered that L3 will take care of most of the caching for their data structures, but in L1 and L2 would only contain immutable data, so there is no need to maintain cache coherency. He envisioned a selector that can be configured to tell the CPU whether the range needs to be handled by the HICAMP controller or the CPU.

Open Mike

Steve Muir (VMware) asked if the approaches discussed can be partially implemented (e.g., implement memory snapshots for just for a part of the memory). Gernot Heiser argued against sacrificing the purity of the model, otherwise the model will never take off. Michael argued that you can use HICAMP as an accelerator, not a replacement for paged virtual memory, so they advocate a hybrid model. Simon Peter argued that for Arrakis they do not advocate a hybrid model, but one could retrofit Arrakis onto KVM, for instance.

Jonas Wagner (EPFL) commented that the discussed hardware models seem to map very well for some workloads, but not for all, and asked whether there are systems with little workload diversity for which these systems would work well. Several attendees gave examples of systems that run dedicated workloads (e.g., OLTP) that could benefit from the proposed hardware changes (e.g., snapshots). Jonas also gave an example for functional languages that could implement reference counting more efficiently in hardware. Gernot agreed that functional languages map very well to a data flow model. Simon also argued that garbage collection also maps very well. Jacob Lorch and Eduard Bugnion suggested that hardware-software co-design is a fascinating area for innovation, but we should not rely only on hardware people to design hardware, otherwise the hardware is hard to exploit. Some examples are hardware that can help do efficient garbage collection and hardware that can efficiently demultiplex. Simon said an open question is what happens if the hardware is not flexible enough at demultiplexing: can a software solution be found?

David Molnar (MSR) pointed out a new piece of hardware that looks interesting: tritium batteries that do not require charging. An open question is how to re-architect the OS assuming such new hardware.

Unconference Results

Summarized by Rik Farrow (rik@usenix.org)

Hardware's Role in System Design

Michael Chan presented the summary of what I thought of as Petro Maniatis' session about the future of CPU and system designs. He pointed out that Intel is swayed by what it expects its biggest customers will want in the future, and what systems researchers want. Software writers want better performance, but also better views of the internal metrics collected by processors. Power consumption is one of Intel's biggest focuses right now, but there are also issues of hardware and software mismatch. For example, Barrelfish relies on cache coherency for inter-core communication, but this works poorly for data structures (or anything larger than six cache lines). Finally, software folks struggle to imagine what will come out of the Intel CPU pipeline five years down the road, the current timeframe for integrating changes in CPUs, and secret by design.

Networking CPU Cores

Jeff Mogul presented a summary of John Ousterhout's unconference session, which was focused on John's desire for a high-speed network that would connect CPU cores and their level 1 caches together with very low latency. The conclusion was that switch designers have already worked on a very similar issue, exchanging packets of data across a switch fabric with very low latency, and that John should talk with the people familiar with these designs. Jeff pointed out that John doesn't want queues, but Jeff said that there must be queues.

Augmented Reality and Mobile Sensors

David Molnar (Microsoft) first thanked Franzi Roesner (U Washington) for helping lead this session. Then he explained what is different in new settings, such as Google Glass and more immersive augmented reality (AR) displays: the input and the output. The input is noisy, sounds and video, and much of it should be private. The output must be controlled, so that malicious apps don't overlay reality with their own version—for example, rewriting a sign. The OS must create a permissions experience and abstractions to control what applications can access which data. We no longer have 2D windows, but 3D volumes. AR makes several existing problems much worse.

There are issues of privacy as well, such as bystander privacy, or places that want a complete ban on video recording, like a gym or a bar in Seattle. There are also man-in-the-middle concerns, such as a government that seeks to collect data on its citizens. David suggested having primacy of the physical space—for example, allowing the owner of a space to zap a camera using an infrared laser. He concluded by saying that there is about a two-year window to deal with this before legislatures start mangling these issues.

Programming Language Approaches to Systems

Edward Yang (Stanford) began by pointing out that programming language and software can be codesigned, and you can even build a language just for yourself. They discussed composition and modularity, the ability to have many languages that can work together. They want incrementalism, which means backward compatibility and no flag days, but also the ability to exclude what doesn't work well. Ed mentioned the difficulty in measuring programmer productivity, and concluded by saying that program languages people should be hired, as they often bring useful insights into projects.

Security

The security unconference group was one of the largest, but the ground covered seemed all-too familiar to me. Deian Stefan (Stanford) presented the summary. The group began by considering a trust model for code integrity, then pondered allowing untrusted code to modify or copy data. They posited that they know how to isolate untrusted code, and that the interesting question is how to share data between sandboxes. They next considered machine learning for security, and whether authentication (actually authorization) should be considered on a scale.

They also considered the role of firewalls in security today, concluding that firewalls provide insufficient protection and that getting them to provide better protection would require a huge amount of user interaction. Plus, firewalls do not protect against internal attackers. They ignored the issue that the attacker who has established a beachhead through the typical spearphishing attack is essentially an insider. This negates having a firewall in almost all of the attacks on organizations seen today.

They finished their session by discussing the role of the user in making security decisions, asking whether they can educate non-power users about security. Restructuring designs that avoid requiring the user to make any security decisions was the final point (and a very good one). My apologies for the editorial comments, and while I only witnessed the end of the discussion, I found myself disturbed by hearing old ground covered while summarizing the notes for the entire session.

Big OLTP: Oxymoron or Impending Crises

Using a graphical reference to Oracle, Peter Bailis (UC Berkeley) began the summary for this session with a question: when will the current tech we use break? Peter said that OLTP follows two common patterns: low mutation rate with many queries, or lots of mutation but few queries. And with devices like Google Glass, there will be both high mutation and lots of queries. Closed-world assumptions about databases will no longer hold, with the source of truth being external to the stream processor. They expect to see OLTP combined with OLAP (analytics), and the challenge will remain providing isolation between queries (ACID).

Big Data Analytics

Byung-Gon Chun presented 13 slides, the most thorough and the longest summary. He began with six slides where the group attempted to define big data, and presented a nice sound bite: the three Vs of Volume, Velocity, and Variety. While volume is clear enough when speaking of big data, and velocity obviously refers to the ability to process that data swiftly, variety means that data may be unstructured.

The group came up with eight areas of interest. The first was low latency, i.e., the ability to work interactively, to recognize significant events in data, and to remain efficient as the volume of data grows. Second was data management, which refers to the issues of data labeling, data format (e.g., HDF5), standardization, provenance, and new data structures. Unified execution is a simple concept: being able to process data on a single box or a scaled-up cluster using the same program. The fourth issue, related to unified execution, is unified programming. Spark and Hive were presented as examples. Workflow management was the fifth issue, the ability to schedule and coordinate a set of related jobs, along with tools for doing this.

Their sixth issue was resource management, which implies at least prioritization or constraints that control how many resources a job can use. While an economic approach was suggested, it was also pointed out that Cosmos, a chargeback scheme, is not working. The seventh issue was accuracy, in the sense that sometimes approximate answers, requiring less processing, are acceptable, and there needs to be the ability to adjust the desired accuracy. The final point was configuration complexity, with Hadoop being used as a bad example, having tens of configuration parameters. What is needed is auto-tuning knobs, where the knobs set desired goals instead of tweaking specific parameters.

Elastic OS

Amit Gupta, who presented a paper about elasticity in operating systems, convened this unconference session to further explore the issue. The participants wondered whether an ElasticOS for generic processes is too broad a goal, but perhaps certain applications, or even threads, would be suitable for elasticizing. Elasticizing may occur for different reasons, even shrinking a process when resource costs go up and expanding when costs go down, and the process could use more resources. In the end, the group concluded that they still need to be convinced.

Verification

Ariel Rabkin (Princeton) organized this session, wrote a summary, but left before he could present it. On his slides, he had written that they now believe that increasingly large artifacts can be verified if the artifact was designed with verifications in mind. Formalization of code design is possible, probably usable, but is only cost-effective for safety-critical code, and not usable yet for Web companies.