Conference Reports

In this issue:

The reports from ICAC '14: 11th International Conference on Autonomic Computing and WiAC '14: 2014 USENIX Women in Advanced Computing Summit are available online: www.usenix.org/publications/login.

USENIX ATC '14: 2014 USENIX Annual Technical Conference

June 19–20, 2014, Philadelphia, PA

Summarized by Daniel J. Dean, Rik Farrow, Cheng Li, Jianchen Shan, Dimitris Skourtis, Lalith Suresh, and Jons-Tobias Wamhoff

Opening Announcements

Summarized by Rik Farrow (rik@usenix.org)

Garth Gibson (CMU) opened the conference by telling us that 245 papers were submitted, a record number. Of these, 49 were short papers. Thirty-six papers got accepted right from the start of reviews, and 11 papers were sent back to authors for revisions, resulting in eight more accepted papers for an 18% acceptance rate. Overall, there were 834 reviews by the 32-person PC, along with 16 additional reviewers. To fit 44 papers into two days, there were no keynotes or talks, just 20-minute paper presentations each crammed into two-hour sessions.

Nickolai Zeldovich (MIT), the co-chair, took over the podium and announced two best paper awards. The first was "In Search of an Understandable Consensus Algorithm," by Diego Ongaro and John Ousterhout, Stanford University. The other best paper award went to "HACK: Hierarchical ACKs for Efficient Wireless Medium Utilization," by Lynne Salameh, Astrit Zhushi, Mark Handley, Kyle Jamieson, and Brad Karp, University College London.

Brian Noble, the president of the USENIX Board, presented two of the three annual awards. Tom Anderson, Mic Bowman, David Culler, Larry Peterson, and Timothy Roscoe received the Software Tools User Group (STUG) award for PlanetLab. Quoting Brian as he made the presentation to Tom Anderson, "PlanetLab enables multiple distributed services to run over a shared, wide-area infrastructure. The PlanetLab software system introduced distributed virtualization (aka 'slicing'), unbundled management (where management services run within their own slices), and chain of responsibility (mediating between slice users and infrastructure owners). The PlanetLab experimental platform consists of 1186 machines at 582 sites that run this software to enable researchers to evaluate ideas in a realistic environment and offer long-running services (e.g., content distribution networks) for real users."

Almost as soon as he had sat down, Tom Anderson was on his way back to the podium to receive the USENIX Lifetime Achievement award, also known as the Flame Award. Again, quoting Brian as he read the text accompanying the award, "Tom receives the USENIX Flame Award for his work on mentoring students and colleagues, constructing educational tools, building research infrastructure, creating new research communities, and communicating his substantial understanding through a comprehensive textbook."

This year's ATC featured a second track, the Best of the Rest, where the authors of award-winning systems papers were invited to present their papers a second time. Eight out of the 11 people invited came to the conference, providing an alternate track.

Big Data

Summarized by Cheng Li (chengli@mpi-sws.org)

ShuffleWatcher: Shuffle-aware Scheduling in Multi-tenant MapReduce Clusters

Faraz Ahmad, Teradata Aster and Purdue University; Srimat T. Chakradhar, NEC Laboratories America; Anand Raghunathan and T. N. Vijaykumar, Purdue University

Faraz Ahmad presented his work on improving the performance of the resource utilization in multi-tenant MapReduce clusters. In these clusters, many users simultaneously submit MapReduce jobs, and these jobs are often running in parallel. The most time-consuming phase is to shuffle data from the finished map tasks to the scheduled reduce tasks. For example, 60% of jobs at Yahoo! and 20% of jobs at Facebook are shuffle-heavy. These jobs have negative impacts on the other jobs since they often run longer and incur high network traffic volume. Many related works tried to ensure fairness among jobs and often focused on how to improve throughput. In this work, the authors wanted to improve latency and throughput without loss of fairness.

Their main solution is to shape and reduce the shuffle traffic. To do so, they designed a tool called ShuffleWatcher, which consists of three different policies. The first policy is network-aware shuffle scheduling (NASS). NASS specifies that shuffle may be delayed if the communication and computation from different concurrent jobs are overlapping. Second, the Shuffle-Aware Reduce Placement (SARP) policy assigns reduce tasks to racks containing more intermediate data to make cross-rack shuffle traffic lower. Third, Shuffle-Aware Map Placement (SAMP) leverages the input data redundancy to locate map tasks to fewer racks to reduce remote map traffic.

To demonstrate the effectiveness of the joint work of these three policies, they evaluated the MapReduce jobs running with their tool and compared them to a baseline policy, the fair scheduler. They deployed all their experiments in Amazon EC2 with 100 nodes, each of which had four virtual cores. The results show improved latency and throughput and reduced cross-rack traffic.

Peng Wu (Huawei) wanted to know more about related and similar work in Google or Facebook. Faraz answered that some previous work also targets improvement in throughput and fairness, but they didn't optimize the map or reduce phases. He also said that he didn't know any public results from either Google or Facebook. The session chair, Anthony Joseph (UCB), asked about the interference introduced by longer jobs. Faraz replied that if the policy allows the jobs to run longer or to use more resources, then that is fine.

Violet: A Storage Stack for IOPS/Capacity Bifurcated Storage Environments

Douglas Santry and Kaladhar Voruganti, NetApp, Inc.

Douglas presented their work on persistent in-memory data structures. He said in-memory computing is important because applications like OLAP and OLTP, and ERP jobs cannot tolerate disk and network latencies. In addition to the in-memory computing, one still needs to store the data (e.g., memory state or transactions) on disk. There are two conventional solutions: (1) using a database to manage data and specifying your own table schema; and (2) designing new data structures and explicitly making state persistent. However, if memory is persistent, then developers don't need to map the memory state back to disk. It would be great if there were a persistence layer that divorced data structure selection and implementation from persistence.

Douglas et al. designed Violet, which introduces the notion of a named heap and replicates updates to memory or to persistent storage at the granularity of a byte. It also automatically enforces ordering and consistent image, and supports snapshot creation. There are two core parts in Violet: (1) a Violet library defines a set of data structures that will be automatically replicated to the memory and stored in disks; (2) Violet exposes to developers an interface which can be used to express memory updates and to group multiple updates into a transaction. To use Violet, developers only have to instrument their code with a few Violet keywords. The experimental results show that the instrumentation overhead is not significant. The asynchronous replication improves throughput numbers. The restore time decreases if the number of machines increases.

Somebody asked for a comparison between levelDB and Violet. Douglas replied that they are completely different, since levelDB is a key-value store. Developers can use Violet to build such a data store. The second question regarded what would happen while restoring data from disk if the virtual address is already taken. Douglas replied that Violet always maps the physical address to the same virtual address.

ELF: Efficient Lightweight Fast Stream Processing at Scale

Liting Hu, Karsten Schwan, Hrishikesh Amur, and Xin Chen, Georgia Institute of Technology

Liting Hu explained that buying things from Amazon involves other applications running to serve micro-promotions, likes, and recommendations. Besides these applications, there will also be data mining, for example, to predict games that will become popular. Liting Hu then displayed data flows for these various applications, where some flows are best processed in batches, others require more frequent processing as streams, and still others, like user queries, require immediate processing and millisecond response times. To provide this level of flexibility, the authors developed ELF.

Typically, data gets collected, using tools like Facebook's Flume or Kafka, into storage systems, like HBase. Instead, ELF runs directly on the Web server tier, skipping the just-mentioned data flow. ELF also uses a many master-many worker structure, with a peer-to-peer overlay using a distributed hash table to locate job masters. Job masters aggregate data from workers, then broadcast answers. ELF provides an SQL interface for programmers for reducing data on masters and workers. ELF also uses aggregation and compressed buffer trees (CBTs). In evaluations, ELF outperforms Storm and Muppet for large windows (more than 30 seconds) because of the ability to flush the CBTs.

Chuck (NEC Labs) asked whether they are running the ELF framework on the Web server itself, and Liting answered that yes, they run ELF as agents on Web servers. Chuck then asked whether they were concerned with fault-tolerance issues. Liting said that, yes, their DHT handled failure by finding new masters quickly. Derek Murray (Microsoft Research) asked how they do the streaming, sliding window for connecting components. Liting answered that the flush operation of CBTs allows them to adjust the size of the window, fetch pre-reduced results. Derek said he had a couple of more questions but would take them offline.

Exploiting Bounded Staleness to Speed Up Big Data Analytics

Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, and Gregory R. Ganger, Carnegie Mellon University; Phillip B. Gibbons, Intel Labs; Garth A. Gibson and Eric P. Xing, Carnegie Mellon University

Henggang presented this work on how to trade data freshness for better performance in big data analytics. Big data like Web pages and documents constitutes a huge amount of data. Analytics often have to perform computation over data iteratively. To speed up the computation, developers normally make iterative jobs run in parallel. However, to ensure the correctness, the sync operation is called periodically, and this pattern slows down the computation. The goal of this work is to reduce the sync overhead.

Henggang described their three core approaches to improve performance. The first approach is called Bulk Synchronous Parallel (BSP), in which every iteration is an epoch, and parallel jobs must synchronize with each other at the end of each

epoch. Arbitrarily-Sized BSP is a variant that allows developers to decide the size of epochs to do the synchronization. The last solution is called Stale Synchronous Parallel (SSP) and is more powerful than the previous two solutions. SSP could tune the number of epochs to call sync and also could allow different jobs to synchronize at different speeds.

They evaluated their approaches by running Gibbs Sampling on LDA jobs in eight machines, with the *NY Times* data sets as input. The results show that performance gets better if staleness increases. SSP performs better than BSP. Regarding convergence, more iterations are required if staleness increases.

The first question concerned the bound on the slowness of stragglers. Henggang replied that they have techniques to try to help stragglers catch up. Some people asked whether they checked the quality of results. He replied that they use the likelihood to measure the quality. In other words, if the likelihood is the same or close, then the generated models are qualified. The last question was about examples. He answered that they built the model to discover possible cancers, and compared the experimental results to a doctor's decision.

Making State Explicit for Imperative Big Data Processing

Raul Castro Fernandez, Imperial College London; Matteo Migliavacca, University of Kent; Evangelia Kalyvianaki, City University London; Peter Pietzuch, Imperial College London

Raul presented work on how to explore parallelism in programs written in imperative languages. He started his talk by showing that it is challenging to make imperative big data processing algorithms run in parallel and be fault tolerant, since the algorithms are stateful. On the other hand, many platforms like MapReduce, Storm, and Spark achieve good performance and manage fault tolerance by assuming there is no mutable state. The downside of using these platforms is that developers must learn new programming models.

This work aims to run legacy Java programs with good performance while tolerating faults. The key idea is to have a stateful data flow graph (SDG). In this graph, there are three elements: state element, data flow, and task element. The state element is often distributed, and has two abstractions: partitioned state and partial state. Partitioned state can be processed by a local task element, and partial state may be accessed by a local or global task element. Additionally, the partitioned state requires application-specific merge logic to resolve conflicts. To figure out which state is partitioned or partial requires programmers to provide annotations. To make the computation tolerate faults, they also implemented asynchronous snapshot creation and distributed recovery mechanisms.

Following his talk, some people asked about the size of their data sets used for experiments. Raul answered that the size of data sets varies from a few GBs to 200 GBs. The second question was about the principles used to specify either partitioned or partial. He answered that programmers need to know.

Virtualization

Summarized by Jianchen Shan (js622@njit.edu)

OSv—Optimizing the Operating System for Virtual Machines

Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov, Cloudius Systems

Nadav Har'El presented their research on OSv, which is a new OS designed specifically for cloud VMs. The goal of OSv is to achieve better application performance than traditional Linux. The OSv is small, quick booted, and not restricted to a specific hypervisor or platform. Nadav Har'El claimed that OSv was actively developed as open source so that it could be a platform for further research on VM OSes. OSv is developed using C++11 and fully supports SMP guests.

Nadav Har'El pointed out that the key design of OSv is to run a single application within a single process, multiple threads within a single address space, because the hypervisor and guest are enough to isolate the application just like the process is isolated in traditional OSes. There is no protection between user space and kernel space such that system calls are just function calls, which means less overhead. Nadav Har'El also emphasized that OSv entirely avoids spinlock by using a lock-free scheduler, sleeping mutex, and paravirtual lock. Basically speaking, there is no spinlock in OSv, so serious problems such as lock holder preemption are avoided. OSv takes advantage of network stack redesign proposed by Van Jacobson in 2006. OSv provides a new Linux API with lower overhead such as zero-copy lockless network APIs. Nadav Har'El suggested that we can improve performance further with new APIs and modifying the runtime environment (JVM), which can benefit all unmodified JVM applications.

Beside the evaluations that can be found in the paper, Nadav Har'El also showed some unreleased experimental results. For the Cassandra stress test (READ, 4 vCPUs, 4 GB RAM), OSv is 34% better. For Tomcat (servlet sending fixed response, 128 concurrent HTTP connections, measure throughput, 4 vCPUs, 3 GB), OSv is 41% better. Finally, Nadav Har'El invited people to join the OSv open source project: http://osv.io/.

Because there is no protection between user space and kernel space, the first questioner worried that if something evil is done in the user space, this could cause some security issues. Nadav Har'El answered that one VM would only run a single application, so only the application itself would be affected, and no damage would be made to other applications or the underlying hypervisor.

Gleaner: Mitigating the Blocked-Waiter Wakeup Problem for Virtualized Multicore Applications

Xiaoning Ding, New Jersey Institute of Technology; Phillip B. Gibbons and Michael A. Kozuch, Intel Labs Pittsburgh; Jianchen Shan, New Jersey Institute of Technology

Xiaoning Ding addressed the Blocked-Waiter Wakeup problem (BWW) and proposed its solution, Gleaner. As the number of

vCPUs in a virtual machine keeps increasing, Xiaoning Ding said that the mismatch between vCPU abstraction and pCPU behavior could prevent applications from effectively taking advantage of more vCPUs in each VM. Xiaoning Ding explained that vCPUs are actually schedulable execution entities, and there are two important features of a vCPU: First, when a vCPU is busy, it may be suspended without notification; second, when vCPU is idle, it needs to be rescheduled to continue computation, not like the physical CPU that can be available immediately for ready computation. For the synchronization-intensive applications that use busy waiting lock, the first feature would cause the Lock Holder Preemption problem (LHP). The vCPU holding the spinlock may be preempted, which makes other vCPU spend a long time waiting for the lock. For those applications that use blocking locks, the second feature would cause the BWW problem: Waking up blocked threads takes a long time on idle vCPUs. Through experiments, Xiaoning Ding showed that waking up a thread on pCPU only takes 8μs, but waking up a thread on vCPU takes longer than 86μs, which is also variable. The major source of overhead and variation comes from the vCPU switch. So that's why BWW would increase execution time and incur unpredictable performance and reduced overall system performance.

Xiaoning Ding pointed out that the LHP problem has been well studied. So their team focused on the solution to the BWW problem. Generally, the goal is to reduce harmful vCPU switching, and there are two main methods to deal with this. First is resource retention: preventing an idle vCPU from being suspended by letting it spin instead of yielding hardware resources, although this may cause resource under-utilization. Second is consolidation scheduling: consolidating busy periods and coalescing idle periods on vCPUs. This method activates some vCPUs to avoid vCPU switching and suspends other vCPUs to save resources. However, the problem is that the active vCPUs may be overloaded. Some active vCPUs may undertake too heavy a workload because some workloads cannot be evenly distributed among active vCPUs. Gleaner basically takes advantage of both of these two methods. At the same time, Gleaner provides a solution to the overloading problem, gradually consolidating workload threads only if the following conditions are satisfied: vCPU utilization would not be too high after consolidation and workloads can be evenly distributed among active vCPUs. Also, Gleaner stops consolidation when the throughput tends to decrease. Xiaoning Ding concluded that the evaluations prove that Gleaner can improve application performance by up to 16x and system throughput by 3x.

Someone was interested in whether Gleaner could also perform well in other hypervisors like Xen. Xiaoning Ding replied that the current evaluation was only done in KVM, so further experiments may need to be done across different hypervisors. Xiaoning Ding was also asked why there is still some slowdown relative to bare-metal performance in the current experimental results. He answered that this slowdown may still be caused by the overloading problem, which cannot be entirely prevented.

HYPERSHELL: A Practical Hypervisor Layer Guest OS Shell for Automated In-VM Management

Yangchun Fu, Junyuan Zeng, and Zhiqiang Lin, The University of Texas at Dallas

Yangchun Fu explained that current cloud services or datacenters usually host tens of thousands of virtual machines, which require large scale and automated management tools. Traditional management tools are placed in the user space of the host OS and work with the management utilities installed in the guest OS. Although the management is centralized, each VM is required to install the client utilities, and the user-space tool also needs the admin password to access the VM, which is painful when dealing with large scale. Hence Yangchun Fu's team proposed the HYPERSHELL, a practical hypervisor layer guest OS shell tool for automated in-VM management, which only installs the management utilities at the hypervisor layer.

Yangchun Fu stated that the system call is the only interface to request OS service, so HYPERSHELL introduces the reverse system call to achieve the guest OS management. The main contribution of the reverse system call is bridging the semantic gap for the hypervisor layer program between the guest and host OS so that the hypervisor can interpret semantic information about the guest OS. The reverse system call technique would dispatch the system call from the host library space program. To differentiate the host and guest system call, Yangchun Fu's solution would add an extra value to the file descriptor, since it is just an index and has a limited maximum value. The system call is arranged to execute by hypervisor. The hypervisor along with a helper process inside the guest OS's user space would inject the transferred guest system call. Yangchun Fu explained that the helper process is created by the hypervisor layer program and would inject the guest system call right before entering the kernel space or exiting to the user space. And because the system call data is saved and exchanged in shared memory, many of the current guest OS management utilities can be directly reused in HYPERSHELL without any modification.

Yangchun Fu said they evaluated about a hundred management utilities and demonstrated that HYPERSHELL has relatively little slowdown and overhead on average compared to their native in-VM execution. More detailed information could be found in the paper. Yangchun Fu concluded that HYPERSHELL will circumvent all existing user logins and the system audit for each managed VM, so it cannot be used for security-critical applications unless special care is taken for these uses. HYPERSHELL requires both OSes running in the host OS and VM to have a compatible system-call interface. But, if further work can be done in additional system-call translation, the HYPERSHELL can work with more kinds of OSes.

Someone asked whether there is any extra overhead when HYPERSHELL is used for monitoring. Yangchun Fu replied that there is no extra overhead.

XvMotion: Unified Virtual Machine Migration over Long Distance

Ali José Mashtizadeh, Stanford University; Min Cai, Gabriel Tarasuk-Levin, and Ricardo Koller, VMware, Inc.; Tal Garfinkel; Sreekanth Setty, VMware, Inc.

Ali José Mashtizadeh explained that previous live virtual machine migration only happened between the machines that shared local shared storage, such as a cluster with a storage array, and only the memory is migrated. But faster networks and current VM deployment, which usually hosts VMs on tens of thousands of physical machines over sites far away from each other, have made necessary a reliable and efficient wide area virtual machine migration technique. Current ad hoc solutions are often complex and fragile, however, so Ali José Mashtizadeh and his team have proposed a solution called XvMotion, which is an integrated memory and storage migration system that does end-to-end migration between two physical hosts over the local or wide area. XvMotion offers performance and reliability comparable to that of a local migration and has been proven to be practical for moving VMs over long distances, such as between California and India, over a period of a year.

Ali José Mashtizadeh said that the architecture of XvMotion contains the live migration and I/O Mirroring modules in ESX. Between the source and destination, the Streams layer handles the large data transfer, including memory pages and disk blocks on top of the TCP network. The live migration module would be responsible for regular tasks as found in local live migration. The I/O Mirroring would record any changes during the copy, which is asynchronously implemented with Streams and the disk buffer. This process can reduce the impact on the source VM from high and unstable network latency. Ali José Mashtizadeh pointed out that in long distance memory migration, if the network transferring rate were slower than the VM's workload changing page rate, then there would be an inconsistency issue and convergence would not occur. In traditional local networks, the VM would be halted and transfer all remaining dirty pages during downtime. This solution is not acceptable, because long distance incurs long latency, which would lead to longer downtime.

XvMotion's solution is Stun During Page Send, which can inject latency in page writes operation to throttle the page dirtying rate to a desired level when it is faster than the network transmit rate. In terms of disk buffer congestion control, Ali José Mashtizadeh refers listeners to the paper for details. The evaluation of XvMotion showed it provides stable migration time and downtime (atomic switchover) less than one second even for latencies as high as 200 ms and data loss up to 0.5%. Downtime only has a small linear time increase with distance. In addition, the guest workload performance penalty is nearly constant with respect to latency, and only varies based on workload intensity.

GPUvm: Why Not Virtualizing GPUs at the Hypervisor?

Yusuke Suzuki, Keio University; Shinpei Kato, Nagoya University; Hiroshi Yamada, Tokyo University of Agriculture and Technology; Kenji Kono, Keio University

Yusuke Suzuki described GPUvm, a technique to fully virtualize GPU at the hypervisor. Currently, GPUs are used not only for graphics but also for massively data-parallel computations. GPGPU applications are now widely accepted for various use cases. However, the current GPU is not virtualized, and we cannot multiplex a physical GPU among virtual machines or consolidate VMs that run GPGPU applications. Yusuke Suzuki said that GPU virtualization is necessary. There are now three virtualization approaches: I/O pass-through, API remoting, and paravirtualization. I/O pass-through assigns a physical GPU to VM dedicatedly. The problem is that multiplexing is impossible. API remoting can multiplex GPUs because it forwards API calls from VMs to the host's GPUs. But API remoting needs host's and VM's API and its version to be compatible. The paravirtualization approach provides an ideal GPU device model to VMs. But each VM uses PV-drivers, which must be modified to follow the ideal GPU device model. The goal of GPUvm is to allow multiple VMs to share a single GPU without any driver modification. At the same time the performance bottlenecks of full virtualization are identified and optimization solutions are accordingly provided in GPUvm.

Yusuke Suzuki explained that there are three major components in a GPU: GPU computing cores, GPU channels, and GPU memory. The GPU channel is a hardware unit to submit commands to GPU computing cores. Memory accesses from computing cores are confined by GPU page tables. GPU and CPU memory spaces are unified. GPU virtual address (GVA) is translated into CPU physical addresses as well as GPU physical addresses (GPA). So the basic idea of GPUvm is to virtualize these three major components. The GPU memory and channels are logically partitioned to virtual ones. And time sharing is employed to make GPU cores shared by several VMs. As a result, each VM would have a set of isolated resources. GPUvm works as a hypervisor, which exposes a virtual GPU device model to each VM. Hence the guest GPU driver needn't be modified.

In GPUvm, the GPU shadow page table isolates GPU memory. GPU shadow channel isolates GPU channels. And GPU fair-share scheduler isolates performance on GPU computing cores. Memory accesses from GPU computing cores are confined by GPU shadow page tables. Since DMA uses GPU virtual addresses, it is also confined by GPU shadow page tables, which guarantees that DMA requested from one VM cannot gain access to CPU memory regions assigned to other VMs. For the GPU channels, mapping tables are maintained to partition the physical channels. Finally, for equitable use of GPU cores, the BAND scheduling algorithm [Kato et al. '12] is used, since the GPU command execution is non-preemptive and BAND is aware of this feature. Their evaluation showed that for long non-preemptive tasks, BAND can provide better fairness compared to some traditional

scheduling algorithms. Yusuke Suzuki said the raw full GPU virtualization may incur large overhead.

Besides describing the architecture of GPUvm, Yusuke Suzuki also introduced some optimization techniques. First, GPUvm allows the direct BAR accesses to the non-sensitive areas from VMs. This reduces the cost of intercepting MMIO operations. Second, GPUvm delays the updates on the shadow table, since this operation is not needed until the channel is active. Third, GPUvm provides a paravirtualized driver to further reduce the overhead of the shadowing table. The evaluation used Gdev (open-source CUDA runtime) and Nouveau (open-source device driver for NVIDIA GPUs) on Xen. The paravirtualized version performed best but is still 2–3x slower than Native execution. Also, GPUvm would incur large overhead in cases involving four and eight VMs without paravirtualization, since page shadowing locks GPU resources.

Yusuke Suzuki and an attendee discussed the pros and cons of GPU hardware virtualization and software virtualization. Yusuke Suzuki pointed out that changing scheduling cannot be supported in hardware virtualization. Someone also suggested methods like killing long tasks to achieve preemption, so that the traditional scheduling algorithm can also provide good fairness in GPU hardware virtualization.

A Full GPU Virtualization Solution with Mediated Pass-Through

Kun Tian, Yaozu Dong, and David Cowperthwaite, Intel Corporation

Kun Tian presented their work on GPU virtualization and began with GPU virtualization's three requirements: native GPU acceleration performance, full features with consistent visual experience, and sharing among multiple virtual machines. Among existing methods, API forwarding can share a single GPU for several VMs, but the overhead and API compatibility problem limits its performance and the features supported. Another method for VMs to use a GPU is direct pass-through. It provides 100% native performance and full features, but no sharing at all. To meet the above three requirements, Kun Tian proposed their team's solution called gVirt, which supports full GPU virtualization and combines mediated pass-through with a performance boost. gVirt can provide full-featured vGPU to VM and avoid VM modifying the native graphics driver. gVirt can also achieve up to 95% native performance and scale up to seven VMs.

Kun Tian said that gVirt is open source and the current implementation is based on Xen, codenamed XenGT, with KVM support coming soon. gVirt supports Intel Processor Graphics built into fourth-generation Intel Core processors. But Kun Tian mentioned that the principles behind gVirt can also apply to different GPUs, since most modern GPUs only have major differences in how graphics memory is implemented. gVirt is trademarked as Intel GVT-g: Intel Graphics Virtualization Technology for virtual GPU. Kun Tian said the main challenges of GPU virtualization are complexity in virtualizing, efficiency when sharing the GPU, and secure isolation among the VMs. In terms of efficiency when sharing the GPU, gVirt takes advantage of mediated pass-through, which passes through performance-critical operations but traps and emulates privileged operations. Trap-and-emulation can provide a full-featured vGPU device model to VMs and use the shadow GPU page table.

In terms of sharing, gVirt partitions the GPU memory and avoids address translation by employing address space ballooning. In terms of secure isolation, among many problems Kun Tian gave an example of the most important one: the vulnerability from direct execution. Although the direct command needs to be audited when using mediated pass-through, after gVirt has received, audited, and submitted the command to the GPU, the GPU may execute the modified command where something evil happens. The solution is smart shadowing: lazy shadowing the statically allocated ring buffer and write protection of the batch buffer, which is allocated on-demand. These two methods can prevent the modification of command's content or prevent the modified command from being executed. At last, Kun Tian said, they found the overhead mainly comes from the power management register's accesses operation, which is unnecessary in VM. So gVirt removed these operations and added more commands submitted in some benchmarks, which led to improved performance.

Someone asked about the progress of the project. Kun Tian said that many more small changes have been included in the newest version. He also provided the publicly available patches for further reference: (1) https://github.com/01org/XenGT-Preview-xen; (2) https://github.com/01org/XenGT-Preview-kernel; (3) https://github.com/01org/XenGT-Preview-qemu.

Best of the Rest I

Summarized by Cheng Li (chenglii@cs.rutgers.edu)

Naiad: A Timely Dataflow System

Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martin Abadi, Microsoft Research

Derek Murray said that a new programming model is required for new applications. A timely dataflow system should be able to handle batch processing, stream processing, and graph processing applications. For example, how would one build a system to handle large social networks that minimizes latency for coordination?

Murray revisited a number of dataflows such as parallelism and iteration. Then he showcased and contrasted the two systems using batching and streaming applications. Batching systems require coordination but need to support aggregation. Streaming systems do not require coordination but aggregation is difficult.

Naiad is different because it is tightly coupled with the execution mode and uses batch processing, stream processing, and graph processing. The timely framework will update results in real-time, minimizing the latency through coordination of events. The timely system supports asynchronous and fine-grained synchronous execution. To achieve low latency, three techniques

were used: a programming model, a distributed progress tracking protocol, and system performance engineering were proposed.

The programming model is event based. Each operation corresponds to an event. Messages are delivered asynchronously. Notifications will support batching. These programming frameworks leverage a timely dataflow API to run programs in a distributed manner. To achieve low latency, Naiad proposes a distributed progress tracking protocol that assigns a timestamp to each event. Sometimes, an event may depend on its own output. So a structured timestamps-in-loops solution was proposed. One challenge of performance engineering in Naiad is to reduce micro-stragglers that negatively impact the overall performance.

The evaluation results showed that Naiad can achieve the design goals with low overheads. Murray demonstrated PageRank, interactive graph analysis, and query latency. His conclusion was that Naiad achieved the performance of specialized frameworks and provided the flexibility of a generic framework.

One question was whether the generic programming model, in providing a fair share in a multi-tenant environment, ran into resource competition problems. Murray answered no. Someone asked how to do fault tolerance. Murray said query and log the state.

Towards Optimization-Safe Systems: Analyzing the Impact of Undefined Behavior

Xi Wang, Nickolai Zeldovich, M. Frans Kaashoek, Armando Solar-Lezama, MIT CSAIL

Xi Wang used an example (digit overflow) to demonstrate that compiler optimization flags may optimize aggressively or discard some of the sanity checks and change the accuracy of the results. Then he used a table to show that it is a common error for widespread GCC versions.

The notion of undefined behavior refers to the behaviors that are not defined by the specifications. The original goal is to emit efficient code, but it allows the compilers to assume a program never invokes undefined behaviors. So this ambiguity causes many undefined behaviors. Therefore, there is a need for a systematic approach to study and control undefined behavior.

The methodology is to use a precise flag to annotate unstable code and compare the two versions with or without the unstable code. A Boolean satisfaction table is used to justify the behavior of two programs when unstable code is enabled or disabled. A framework, STACK, is proposed to identify unstable code. STACK makes the compute assumption as no undefined behavior. Then STACK will run the two versions and compare the diffs. The limitation is missing unstable code and false warnings. The presenter addressed these issues separately.

The evaluation shows STACK is robust to find unstable code and scales to large code bases. The presenter also made suggestions on how to avoid unstable code.

Storage

Summarized by Daniel J. Dean (djdean2@ncsu.edu)

vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment

Fei Meng, North Carolina State University; Li Zhou, Facebook; Xiaosong Ma, North Carolina State University and Qatar Computing Research Institute; Sandeep Uttamchandani, VMware Inc.; Deng Liu, Twitter

Fei Meng began by describing how SFC can be used to accelerate I/O performance, reduce I/O disk load, and reduce contention. Fei then discussed how current SFC management techniques, including static partitioning and globally shared SFC space. Both have issues. To address these challenges, Fei introduced vCacheShare, which can dynamically resize the cache as needed. He also described how vCacheShare takes both the long term and short term into account when deciding the cache size. The four modules of vCacheShare are: (1) a cache module, responsible for cache management; (2) a monitor, responsible for tracing cache usage; (3) an analyzer, responsible for analyzing cache usage based on the traces from the monitor; and (4) an optimizer, responsible for optimizing cache usage. Finally, Fei discussed the results of micro- and macrobenchmarks they ran that demonstrated the effectiveness of their tool.

The first question was whether the number of VMs affects the system. Fei answered that the number of VMs does not affect the system. Someone else asked whether context switches affect the system, to which Fei answered they do not.

Missive: Fast Application Launch From an Untrusted Buffer Cache

Jon Howell, Jeremy Elson, Bryan Parno, and John R. Douceur, Microsoft Research

In this talk, Jon Howell described how to quickly launch relatively large applications from an untrusted buffer cache. The authors based their work on the Embassies system, which uses an ultra-lightweight client to ensure applications are isolated and self-contained. By making the client small, however, a large amount of application data will need to be sent over the network, causing applications to take a long time to launch.

To address this issue, Jon described how they collected and analyzed 100 "best-of" applications in order to find ways to speed up the launch process. They found that there was a lot of commonality among the applications, which they could exploit to launch applications faster. Specifically, they used zar files and Merkle trees to only send what was absolutely necessary to launch the application in an efficient way. Some of the details Jon emphasized were how zarfile packing works. Specifically, large files are packed first and smaller files are packed in between, leading to very little wasted space. By using this approach, Jon then showed how it was possible to launch relatively large applications in hundreds of milliseconds.

Someone asked whether they tried to optimize anything. Jon answered that they had not yet optimized anything and wanted to first show a proof that the idea could be done.

A Modular and Efficient Past State System for Berkeley DB

Ross Shaull, NuoDB; Liuba Shrira, Brandeis University; Barbara Liskov, MIT/CSAIL

Ross Shaull described Retro, a system for snapshotting and past state analysis for Berkeley DB. He began by describing why it is useful to know the past states of the DB, using several examples such as auditing, trend analysis, and anomaly detection. The problem is that saving past states is difficult, and not all data stores provide the ability to do it. Ross then described Retro, a low overhead snapshot system for Berkeley DB.

Ross discussed how Retro was designed with simplicity in mind, making choices such as extending existing Berkeley DB protocols instead of creating new protocols. He then described how Retro snapshots are consistent, global, named, and application-declared. Retro works by tagging which pages to save, which pages are part of a query, and which pages to recover when a crash occurs. Ross also described how recovery with Retro works. Specifically, Retro saves pre-states during Berkeley DB recovery, then reads a page from disk and applies redo records to it. Finally, Ross showed that Retro is non-disruptive, imposing 4% throughput overhead. He also conceded, however, that Retro does require a separate disk for good performance.

The first question was how has this advanced the state of the art. Ross replied that the past-state system is integrated into a different system layer, which can be inserted into the database near the page cache.

SCFS: A Shared Cloud-backed File Systems

Alysson Bessani, Ricardo Mendes, Tiago Oliveira, and Nuno Neves, Faculdade de Ciências and LaSIGE; Miguel Correia, INESC-ID and Instituto Superior Técnico, University of Lisbon; Marcelo Pasin, Université de Neuchâtel; Paulo Verissimo, Faculdade de Ciências and LaSIGE

Alysson Bessani described SCFS, a shared cloud-backed file system that is designed to provide reliability and durability guarantees currently lacking in existing systems. Alysson began by describing the two main classes of shared storage: local software, which interacts with a backend (e.g., Dropbox), and direct access-based systems (e.g., BlueSky). While these systems work well, they do not make any reliability or durability guarantees and instead offer a best effort approach.

To address this challenge, the authors designed SCFS. Alysson discussed a key idea behind their approach: always write and avoid reading; in cloud-backed storage systems, writes are essentially free while reading is typically more expensive. To do this, they use a consistency anchor in order to make sure everything done locally is consistent with whatever is in the cloud. They use a consistency service to ensure the correct version of a file is obtained initially, they then perform all operations on the file locally, and finally push it to the cloud when the file is closed. Another point Alysson discussed was how SCFS can use multiple backends in order to ensure data is available even when faced with data-corruption or service unavailability. Finally, the experiments they conducted demonstrated the pros and cons of their approach under various modes of operation (e.g., blocking vs. non-blocking).

Someone asked whether they compared FS Cache to their work. Bessani answered that the files are only sent to the cloud when calls close. The questioner also wondered what files were cached, to which Bessani answered that all locally modified files were cached, with the master version being on the cloud.

Accelerating Restore and Garbage Collection in Deduplication-based Backup Systems via Exploiting Historical Information

Min Fu, Dan Feng, and Yu Hua, Huazhong University of Science and Technology; Xubin He, Virginia Commonwealth University; Zuoning Chen, National Engineering Research Center for Parallel Computer; Wen Xia, Fangting Huang, and Qing Liu, Huazhong University of Science and Technology

Min Fu described how to accelerate the restore and garbage collection process in deduplication-based systems. He began by describing how fragmentation is a major problem for these systems and how it can negatively affect garbage collection and restoration. Specifically, Min discussed how sparse and out-of-order containers cause problems.

To address this problem they have developed a history-aware rewriting algorithm that reduces sparse containers. Min said that the idea of taking history into account is based on the fact that two consecutive backups are very similar; the data contained in the previous backup is useful for the following backup. Min then described two optimization approaches to their new algorithm that can reduce the negative impact of out-of-order containers. Finally, the results shown demonstrated the effectiveness of their approach in terms of performance versus various commonly used approaches.

The first question was how the merge operation would be handled by their garbage collection algorithm. The answer Min gave was that they don't need to merge with their scheme.

Hardware and Low-level Techniques

Summarized by Jons-Tobias Wamhoff (jons@inf.tu-dresden.de)

The TURBO Diaries: Application-controlled Frequency Scaling Explained

Jons-Tobias Wamhoff, Stephan Diestelhorst, and Christof Fetzer, Technische Universität Dresden; Patrick Marlier and Pascal Felber, Université de Neuchâtel; Dave Dice, Oracle Labs

Jons-Tobias Wamhoff proposed that multithreaded applications should gain control over the frequency of the underlying processor cores such that they can expose their possibly asymmetric properties and improve performance. Traditionally, dynamic voltage and frequency scaling (DVFS) is used to save energy by reducing the voltage and frequency if there is only low load on the system and to boost a subset of the cores to speed up sequential bottlenecks or peak loads. Instead of relying on the transparent solution by the operating system and processor, he introduced a user-space library that allows programmatical control of DVFS.

In his talk, Jons-Tobias first gave an overview of DVFS implementations on current AMD and Intel x86 multicores and a

study of their properties regarding frequency transition latencies and power implications. The study shows when frequency scaling improves the efficiency and uses a benchmark that continuously tries to execute critical sections on all cores. The results indicate that blocking locks using the futex syscall are preferred over spinning locks if the critical section has a length of at least 1.5M cycles (500µs) to outweigh the overhead of halting the cores and boosting the frequency. With manual DVFS control, all cores can remain active at a low voltage while one core can boost its frequency. This allows reducing the break-even point of DVFS to boosted sections with at least 200k cycles (50µs). After presenting the DVFS cost, an overview of the TURBO library followed as well as a teaser for the use cases in the paper that apply the library to real-world applications.

After the talk, someone asked how frequency scaling is limited by accesses to the last level cache. Jons-Tobias replied that the L3 cache is not part of the core's frequency domain and can limit the performance when the instructions per cycle depend on the core frequency.

Implementing a Leading Loads Performance Predictor on Commodity Processors

Bo Su, National University of Defense Technology; Joseph L. Greathouse, Junli Gu, and Michael Boyer, AMD Research; Li Shen and Zhiying Wang, National University of Defense Technology

Joseph L. Greathouse started his presentation by asking how fast applications will run at different CPU frequencies. He highlighted that applications are affected by DRAM accesses, which hinder the scaling of performance to higher frequencies. Therefore, a good estimate of the memory time is required to be able to reason about the speed of the remaining CPU time. Many estimates are based on the last level cache misses, but those are not a good indicator because memory accesses can be processed in parallel. In such situations, the CPU time overlaps with the memory accesses and can still scale with the frequency.

The proposed solution focuses on leading loads, which are the first in a series of parallel cache misses to leave the CPU core. The remainder of the talk explained how leading loads can be estimated using existing performance counters with an average error of 2.7%. AMD processors maintain a miss address buffer, a list of L2 cache misses that will be served from the L3 cache or memory in the order they were requested. The event of a new entry in the first provision of the buffer demarks a new parallel memory access period. A hardware performance counter makes the event available. Together with the timestamp counter, this can be used to estimate the portion of the execution time that is spent waiting for data that is not available within the frequency domain (core, L1 & L2 cache).

Afterwards, someone noted that (1) the error is only slightly affected by the frequency, (2) no matching performance counter on Intel processors is known, and (3) the impact of the memory throughput becomes visible if the memory runs out of bandwidth and the latency increases.

HaPPy: Hyperthread-aware Power Profiling Dynamically

Yan Zhai, University of Wisconsin; Xiao Zhang and Stephane Eranian, Google Inc.; Lingjia Tang and Jason Mars, University of Michigan

Yan Zhai addressed the power accounting on servers at individual job granularity with the goal of allowing billing based on power and power capping. The focus of power accounting is on the processor because it is responsible for the biggest part of the power draw. Unfortunately, the simple approach of estimating the power draw linearly to the CPU usage does not work for hyperthreading systems because the processor cores are a shared resource.

The talk introduced a hyperthreads-aware power profiler, which first maps the socket power to the processor cores and then from the core's power to the hyperthreads. The solution is based on finding a factor that gives a ratio of the core's power to the active hyperthreads. This is done by weighting the cycles: The cycles for each hyperthread are captured and used to map the core's power to the hyperthreads. The approach allows reducing the prediction error to 7.5%.

After the talk, Yan Zhai clarified that the approach also works for power cores in large SMT systems and that finding the factor is based on samples that allow an adaption when the load or the application characteristics change.

Scalable Read-mostly Synchronization Using Passive Reader-Writer Locks

Ran Liu, Fudan University and Shanghai Jiao Tong University; Heng Zhang and Haibo Chen, Shanghai Jiao Tong University

Ran Liu started the talk with an overview of the synchronization evolutions to highlight that the mechanisms trade semantic guarantees for performance. By allowing readers and a writer to proceed in parallel, RCU removes the reader side memory barrier. Prior research reveals that the efficiency of reader-dominant synchronization would improve significantly if no barrier was required on the reader side. However, RCU adds considerable constraints to programming due to its weaker semantic compared to reader-writer locks.

While active locks keep the state always consistent using adequate barriers, passive reader-writer locks remove the barriers on the reader side by making the state only consistent if the writer becomes active. However, the writer can only wait for the readers to report their state but it cannot directly check it. The period a writer has to wait is bounded by enforcing the readers to report using IPI. The algorithm only works if TSO is guaranteed because it implicitly enforces that readers see the latest state from the writer. Ran Liu reported that passive reader-writer locks can be implemented and applied to the address space management in Linux with trivial effort. The evaluation showed scalable results similar to RCU while maintaining the reader-writer locks semantic.

There was no time for questions at the end.

Large Pages May Be Harmful on NUMA Systems

Fabien Gaud, Simon Fraser University; Baptiste Lepers, CNRS; Jeremie Decouchant, Grenoble University; Justin Funston and Alexandra Fedorova, Simon Fraser University; Vivien Quéma, Grenoble INP

Baptiste Lepers' talk was about the efficiency of large pages on NUMA systems. Since a TLB miss is expensive (43 cycles), applications that need large amounts of memory typically increase the page size from 4 KB to 2 MB to reduce the address translation overhead and have fewer TLB misses. Unfortunately, large pages may lead to a bad placement of memory on a NUMA system and hurt the performance by up to 43%. Existing memory management algorithms are not able to solve the performance decrease. The talk identifies two effects that explain bad performance: (1) hot pages, i.e., a single page that concentrates most of the memory accesses and creates contention (such a page is far more likely with 2 MB pages than with 4 KB pages); and (2) "page level false sharing," i.e., when different threads allocate distinct data that unfortunately end up being allocated on the same page—this is bad for locality if the two threads are on different nodes. These two effects can lead to a bad locality and high contention on the interconnect.

The performance bottlenecks are addressed by (1) splitting hot pages, (2) improving the locality by migrating pages to the core that accesses it most frequently, and (3) enabling 2 M pages only if the TLB miss rate is very high. The evaluation showed that the approach can lead to a performance improvement of up to 50% while introducing only 3% overhead.

Someone asked if this works for datacenter-scale applications. Lepers answered by stating the evaluation included benchmarks that use up to 20 GB memory but showed sometimes mixed results.

Efficient Tracing of Cold Code via Bias-Free Sampling

Baris Kasikci, École Polytechnique Fédérale de Lausanne (EPFL); Thomas Ball, Microsoft; George Candea, École Polytechnique Fédérale de Lausanne (EPFL); John Erickson and Madanlal Musuvathi, Microsoft

Baris Kasikci said his team's goal is to efficiently sample cold code because such code is not known a priori and is typically not well tested. Existing code instrumentation techniques are inefficient or do not scale: static instrumentation has high overheads and existing dynamic instrumentation frameworks do not work well for multithreaded applications, because they need to stall all program threads before instrumenting the program.

The proposed solution is based on leveraging breakpoints. The code is assigned one breakpoint per basic block that can be removed when the block is sampled, incurring no subsequent overhead. Insertion and deletion of a breakpoint are atomic in modern hardware and hence do not require synchronization. This way, there is no need for a separate program build (easier maintenance), and threads are better supported. The remaining challenge is the effective and efficient handling of the high volume of breakpoints that fire.

The bias-free sampling approach samples code independent of the execution frequency of its individual instructions, and it takes only a specific number of samples from all basic blocks. This way, tasks such as measuring code coverage or periodically sampling instructions can be performed with an overhead of only 1–6%.

During the discussion, Kasikci clarified that it is necessary to stall the threads when inserting or removing multi-shot breakpoints.

Distributed Systems

Summarized by Rik Farrow (rik@usenix.org)

Gestalt: Fast, Unified Fault Localization for Networked Systems

Radhika Niranjan Mysore, Google; Ratul Mahajan, Microsoft Research; Amin Vahdat, Google; George Varghese, Microsoft Research

Radhika Mysore worked on Gestalt when she was a grad student at UCSD. They used Lync, an enterprise communication system within MS that already has a monitoring infrastructure as one data source. But Lync needed an automated fault localization system, because manual localization took hours or days. They started with three existing tools: Score, Pinpoint, and Sherlock, but the diagnostic ranks of Score were terrible, better for Pinpoint, while Sherlock had a good diagnostic rank but took too long to complete. They also tried the same algorithms on an Exchange installation and found that Score was both extremely fast and very accurate. Sherlock was as accurate but still very slow.

Their first contribution was to establish a framework to explain why different algorithms behave differently for different systems, and they built Gestalt based on this framework. For comparing different algorithms, they found that each had a model of system operation, a state space explorer to generate root-cause hypotheses, and finally a scoring function for choosing the most likely causes. Radhika then explained models as ways of encoding systems organization: for example, a deterministic model that is a graph where all edges are equally likely, and a probabilistic model where probabilities are assigned to each edge. Given a model, the state space explorer traverses the model in an attempt to determine which edges may have lead to a failure. Then the scoring function chooses the most likely root-cause.

Radhika focused on one aspect that compounds successful localization: observation noise, or information that falsely reports success or failure of a transaction. This aspect helps to explain why certain algorithms performed poorly. When greedy set cover is used as a space explorer, it performs well when faced with noise, but is slow. Radhika then described how greedy set cover failed when there is noise with an example. Gestalt works better by including a noise factor—and by expecting the real culprit will explain noise—time observations, and fewer observations, and that is how Gestalt achieves better recall. Gestalt performed with high accuracy and low diagnostic time in their testing with data from real systems.

Steve Neil (Comcast) asked, if noise is critical, whether that is something that they determined by looking at all the data they

had. Radhika replied that they looked at a history of failures, looked at these observations, compared these observations with actual data, and came up with the actual value of noise. In addition, she chose noise for the presentation, when there were actually five issues that can confuse automatic fault detection. Neil asked whether this is automated. Radhika responded yes.

Insight: In-situ Online Service Failure Path Inference in Production Computing Infrastructures

Hiep Nguyen, Daniel J. Dean, Kamal Kc, and Xiaohui Gu, North Carolina State University

Hiep Nguyen explained that they examined an online service, a VCL lab with 8000 users, similar to EC2, where students can make requests for VMs. They focused their study on the reservations servers. There were many non-crashing failures in online services, as many as 1813 in one year, which often go unnoticed, like HTTP server threads dying. On top of this, replicating these failures offline is difficult because they are lacking the correct environment, don't want to use record and play, or perform diagnosis directly on a production server. But production environments provide lots of clues—inputs, configuration, logs, and system call traces—that they can use to limit search scope. Finally, they can use dynamic VM cloning to create shadow components so that they can diagnose failure on non-production systems.

Analysis is still difficult because they are using a binary-based approach, want to have low overhead (no intrusive recording), and are analyzing both compiled and interpreted programs. Their solution is to use guided binary execution exploration, which leverages the production environment data and runtime outputs as guidance to search the potential failure paths. They allow the dynamically created clone to receive input data and perform reads, but no writes, to prevent side effects on the production system. The guided binary execution exploration combines both input and console logs as constraints in the search for the cause of a fault. They also include system call records to guide the search. Their existing implementation currently supports Perl and C/C++ programs, but with a modified Perl interpreter, and uses the Pin tool for C/C++ programs. Combining all three (input, console log, and system calls) provides the best method to uncover the root cause of faults.

There were no questions.

Automating the Choice of Consistency Levels in Replicated Systems

Cheng Li, Max Planck Institute for Software Systems (MPI-SWS); Joao Leitão, NOVA University of Lisbon/CITI/NOVA-LINCS; Allen Clement, Max Planck Institute for Software Systems (MPI-SWS); Nuno Preguiça and Rodrigo Rodrigues, NOVA University of Lisbon/CITI/NOVA-LINCS; Viktor Vafeiadis, Max Planck Institute for Software Systems (MPI-SWS)

Cheng Li began by saying that developers often choose to use replication to speed up performance, but then need to decide when strong consistency, which weakens performance, is required. Three years ago, they wrote RedBlue consistency (OSDI '12), which builds replicated systems that are fast and correct. Blue means local and fast but with weak consistency, and red operations are globally slow but strongly consistent. To choose between red and blue, you choose red for operations that are not commutative and may break invariants, or you can use the faster blue. You want to maximize the blue state by encoding the side effects. Cheng Li used the example of making deposits and receiving interest in parallel. They created a tool, called Sieve, which classifies side effects into fast/weak and strong/slow operations.

They examined several example applications and noticed that most are divided into two tiers, the application servers and the database. They decided to use commutative replicated data types (CRDT). They transformed each database statement into one or more database transactions. Programmers only need to annotate the schema with a CRDT annotation to have encoded side effects into shadow operations. Cheng Li next explained how to classify operations accurately and efficiently. Sieve statically defines the weakest precondition for the corresponding shadow operation to be invariant preserving. At run time, Sieve classifies shadow operations by evaluating the corresponding weakest precondition. By using path analysis, they can determine which paths might lead to invariants by creating templates.

The programmer's notations guide the path analysis. When evaluated, Sieve using programmer annotations was almost as accurate as manually choosing weak and strong consistency issues, and incurred a very small hit to performance.

Someone from Google asked how much memory their technique added, and Cheng Li answered that they hadn't checked that. Someone else asked about selecting which types of CRDT they should use. Cheng Li answered that they have a table about how to choose CRDT types, as that research had already been done. There was a third questioner who was cut off by the session chair.

Sirius: Distributing and Coordinating Application Reference Data

Michael Bevilacqua-Linn, Maulan Byron, Peter Cline, Jon Moore, and Steve Muir, Comcast Cable

Jon Moore began by explaining that reference data means a read-only relationship with the data, and also that the rate of update to the data is not very high. For Comcast, reference data means TV and movie metadata, and, with main memory capacity growing, the authors hoped to be able to fit all the reference data into memory. But there is an impedance issue, because the application wants the data and the data is stored in a database. Object-relational mappers can be used to perform the conversion, and then application developers are dealing with data structures, algorithms, unit tests, and profilers.

They use the system of reference to publish updates to the version stored in RAM, which does not have to be that fresh. They created Sirius to do this, with just two operations, put and delete, using Paxos for accuracy and a transaction log for persistence. The application, rather than the database, handles these updates. On the read path, applications read directly from

the data structures in RAM, meaning that they have eventual consistency. They have a set of ingest servers, and client servers just pull updates from the ingest servers. Since the data includes a version, they use compaction to compress past transactions to the most recent value for each key. They use Scala and work from a paper (Paxos made moderately complex) to implement this; Jon displayed the code they used.

Sirius is currently being used at Comcast and has been running in production for almost two years. Since they want their programmers focused on the user experience, not the plumbing, this has been very important for them. The library handles persistence and replay. Sirius is available as open source at comcast.github.io/sirius.

Fred Douglis (EMC) asked about related work and Jon answered that the paper does include a long related-work section. As good engineers, they wanted to build on the shoulders of giants. There is a lot of related work, but most is in external processes. They did look around, but a key difference is not just holding data in memory, but rather the convenience to developers. Dave Presotto (Google) wondered why they ended up with Paxos, given their two-layer structure. Jon replied that their work predates Raft [the next paper], and they were looking at lots of work on distributed databases. Paxos was also easy to reason about. Someone pointed out that given their constraints, it was easy to see how they came up with this design. But with a higher update rate, this wouldn't be applicable. Jon replied that he absolutely agreed with the questioner.

In Search of an Understandable Consensus Algorithm

Diego Ongaro and John Ousterhout, Stanford University

Awarded Best Paper!

Diego Ongaro presented Raft as a replacement for Paxos. Consensus requires an agreement about shared state, and while Paxos is synonymous with consensus, it is also hard to understand and hard to implement. When they tested CS students, most tested better using Raft and would prefer to use it.

Raft is broken into three parts: leader election, log replication, and safety. The leader takes commands from clients and appends them to its log, then the leader replicates its logs to other servers. The leader sends out RPCs with updates and gets back replies from clients. The leader gets elected when the previous leader times out. Each server uses a randomized timeout to prevent split votes on leader elections.

The leader's job is to send out logs, and clients maintain two indices: match and next indices. The next index is where the next update goes, and match index is the latest update. The leader doesn't mark an update committed until a majority has responded to a log replication update. If a client server has old data, it accepts updates to overwrite that old data from the current leader. When a candidate server starts an election, but has an out-of-date log, no other servers will vote for it. So safety means the server with the latest log will always win the election.

Consensus is widely regarded as difficult, and Raft is easier to teach in classrooms, with dozens of implementations available on their Web site or at raftconsensus.github.io.

Fred Douglis said that this was a great talk and should have won best presentation, too. Diego pointed out that they could watch the student study on YouTube. Fred then asked if you could wind up with more divisions and split votes. Diego replied that you might need to scale the timeouts to be wider, to prevent split votes from occurring. Someone noted that for reverting logs, they must keep a lot of version information, and wondered whether this created a lot of overhead compared to Paxos. Diego responded that it depends on which Paxos variant you use, but the overhead is comparable. Someone else asked about log compaction, and Diego said that Raft uses a snapshotting approach; it doesn't snapshot the tail, just committed prefixes. Garth Gibson (CMU) wondered why they hadn't tried using Emulab for large-scale testing. Diego replied that the motivation for his work was RAMCloud, which is why he wasn't focused on wide area issues. In the worse case, they may have to change the leader algorithm. Garth Gibson responded that in the worse case, timeouts mean not making progress. Diego replied that Paxos takes 10 message types to do the same thing. A questioner wondered whether they had formalized the algorithm to prove its correctness. Diego said he had, and there was more work ongoing.

Networking

Summarized by Lalith Suresh (lsuresh@inet.tu-berlin.de)

GASPP: A GPU-Accelerated Stateful Packet Processing Framework

Giorgos Vasiliadis and Lazaros Koromilas, FORTH-ICS; Michalis Polychronakis, Columbia University; Sotiris Ioannidis, FORTH-ICS

Giorgos Vasiliadis presented GASPP, a framework for leveraging GPUs to perform network traffic processing. The premise of the work is that network packet processing is both computationally and memory intensive, with enough room for data parallelism. However, this presents many challenges, such as the fact that the nature of traffic processing presents poor temporal locality since each packet is mostly processed only once.

GASPP presents a modular and flexible approach to expressing a broad range of packet-processing operations to be executed on the GPU. It presents a purely-GPU-based technique for flow state management and TCP stream reconstruction. Since real traffic is very dynamic with different rates and varying packet sizes within and across flows, it becomes a challenge to ensure that GPU threads are sufficiently load balanced and occupied. Thus, GASPP also implements an efficient packet-scheduling mechanism to keep GPU occupancy high. In the evaluation, the authors present the tradeoff between latency and throughput, which is inherent to the design of GASPP.

Steve Muir (Comcast) asked whether the authors performed any comparisons versus Packet Shader. Giorgos answered that they haven't done so yet, but expect GASPP to outperform

Packet Shader. Someone from IBM Research asked how Intel's DPDK compares to GASPP. Another author answered that the main difference is that Intel DPDK uses polling and keeps the CPU cores utilized (potentially up to 100%) with the benefit of low-latency packet processing, whereas GASPP offloads packet processing to the GPU. Garth Gibson (CMU) asked how fast the CPU implementation used as a baseline was, and whether any low-level hardware features were used. Giorgos responded that only a single CPU core was used for the CPU-based baseline, but the point they wanted to make was to use both the CPU and GPU together to perform packet processing.

Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks

Dan Levin, Technische Universität Berlin; Marco Canini, Université catholique de Louvain; Stefan Schmid, Technische Universität Berlin and Telekom Innovation Labs; Fabian Schaffert and Anja Feldmann, Technische Universität Berlin

Dan Levin presented Panopticon, an architecture to allow operators to reap the benefits of software-defined networking (SDN) without having to replace all of their legacy network switches with SDN-enabled switches. He argued that while SDN presents operators with a plethora of benefits, performing a full fork-lift upgrade to SDN is impractical for most network operators.

Panopticon thus presents operators with a solution that allows them to incrementally upgrade a network to a partial-SDN network, and then treat this as a logical SDN. The gist of the idea is to replace some legacy switches with SDN switches and then have all the traffic in the network cross at least one SDN switch using VLANs, which then presents a vantage point for controlling the network as an SDN. The Panopticon planning tool takes as input the network topology, estimates of how much traffic is to flow through the network, and performance constraints (such as bandwidth requirements). It applies these inputs against a planning strategy, and then provides an output hybrid SDN deployment that satisfies the necessary constraints. The authors evaluated the work through simulations, emulations, and a real testbed. Their simulations run against the topology of a large enterprise network demonstrate that with upgrading as few as 10% of distribution switches to SDN-capable switches, most of the considered enterprise network can be operated as a single logical SDN.

Steve Muir (Comcast) asked how one positions Panopticon with respect to the trend towards the use of SDN overlays and soft-switch technologies such as OVS and VXLAN. Dan answered that if and when a network can be managed as an SDN by deploying soft switches on hypervisors at servers, then it should be done. But he repeated the point that there are many legacy networks where that cannot be done, as their survey of enterprise networks has shown, and what they are stressing in their work is how to reason about the network during the transition phase to an SDN. Nick Feamster (Georgia Tech) asked how related is the underlying physical network to the logical network when attempting to assert different guarantees. Dan answered that that is indeed a challenge and it is difficult to make strong guarantees when presented with a logical SDN as in Panopticon, which is why they say they can reap the benefits of a nearly full SDN as opposed to a full SDN.

Pythia: Diagnosing Performance Problems in Wide Area Providers

Partha Kanuparthy, Yahoo Labs; Constantine Dovrolis, Georgia Institute of Technology

Partha Kanuparthy presented Pythia, a system for automatic and real-time diagnosis of performance problems in wide area providers, which. Wide area providers are a set of sites that are deployed in different geographic regions connected by wide area links (such as ISPs or content providers). In practice, they are connected by wide area paths with transient providers, which are essentially black boxes. In these scenarios, network upgrades or changes to the traffic matrix can introduce performance problems.

Wide area providers use monitoring infrastructure that essentially runs measurements (such as ping). This infrastructure can provide a time series of end-to-end measurements of delays, packet reorderings, and network paths being used. Pythia, leverages such infrastructure for near real-time network diagnosis, problem detection, and localization by having lightweight agents run at the different monitors in the network. At the heart of Pythia is a pathology-specification language, which would allow operators to add and remove definitions of what constitutes a performance problem, allowing incremental diagnosis deployment. Each pathology is expressed as a mapping of an observation to a logical expression constituting a set of symptoms. Using this specification, Pythia generates diagnosis code as a forest of decision trees. The system then matches the recorded observations from the monitoring infrastructure (such as delays, losses, reorderings) against the decision trees in order to diagnose any detected problems, wherein a problem is defined as a significant deviation from a baseline. An interesting discussion was also presented on how bugs with the monitors themselves complicate the diagnosis of short-lived problems, since the measurements themselves are potentially contaminated.

There were no questions after the talk.

BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks

Srikanth Sundaresan, Sam Burnett, and Nick Feamster, Georgia Institute of Technology; Walter de Donato, University of Naples Federico II

Sam Burnett presented their experience deploying BISmark, a world-wide measurement infrastructure comprising modified home routers. The objective of this infrastructure was to study questions regarding whether or not ISPs are performing as advertised, how home network usage varies across different parts of the world, and how network troubleshooting can be improved. This presents many challenges, since studying home networks is difficult because of network address translation, the fact that these networks are largely unmanaged and unmoni-

tored, and, lastly, the need to involve actual home users in order to deploy such an infrastructure.

To reliably and consistently measure how fast an ISP is or how a wireless access point affects a user's performance, the home router was a natural candidate to be a vantage point. This allowed the authors to study what traffic was coming from the home network, what devices were causing this traffic (mobile phones, home entertainment devices, laptops?), and so understand where the bottlenecks were. BISmark home routers were deployed in more than 30 countries. Sam then discussed the practical challenges involved in making such a project work, since these home routers are on the critical path of a user's network. This challenge extends to issues regarding automatic updates to the software on the routers and ensuring that the routers are running even when the research team cannot access the router. Although the advantage is that the home router is an ideal vantage point for the kind of problems the authors set out to study, there are disadvantages as well. Users naturally have privacy concerns. Furthermore, establishing trust also proved to be difficult. Sam lastly also discussed interesting statistics regarding the deployment effort, including attrition rates, the time it took for each user to turn on their router after receiving it, and so forth.

Steve Muir (Comcast) asked the degree to which path measurements would have sufficed to study the kind of problems the BISmark project was targeting. Sam answered that that there are classes of problems which you cannot see using path measurements. Someone asked whether there was any relationship between the discussed trust issues and human behaviors that can be inferred from studying the usage patterns. Sam acknowledged that as an issue.

Programmatic Orchestration of WiFi Networks

Julius Schulz-Zander, Lalith Suresh, Nadi Sarrar, and Anja Feldmann, Technische Universität Berlin; Thomas Hühn, DAI-Labor and Technische Universität Berlin; Ruben Merz, Swisscom

Julius Schulz-Zander discussed Odin, a software-defined networking (SDN) framework for WiFi. The motivation for the work is that most of the benefits of SDN have been geared towards wired networks and have not benefitted WiFi as much, whereas WiFi networks are becoming increasingly more complex to manage. In order to design an SDN for WiFi and to programmatically manage WiFi networks, new abstractions need to be designed.

Core to how Odin functions is the light-virtual access point (LVAP) abstraction. An LVAP is a per-client virtual access point, which is spawned on physical access points. Every client is assigned an LVAP when it attempts to connect to the network. LVAPs can be migrated quickly between physical access points such that clients do not have to reassociate to the network and do not notice any breakage at the link layer. Using this mechanism, an Odin-managed network can control clients' attachment points to the network. A logically centralized controller manages LVAPs and exposes the programming API with which network applications can orchestrate the underlying network. Using LVAPs, a network can be divided into slices, where each slice is defined as a set of physical APs, network names, and network applications that operate upon it. An application can control only those clients that are connected to the network names of the slice that it belongs to, and thus, applications cannot control LVAPs from another slice. Using these building blocks, the authors built six typical enterprise services on top of Odin.

Nick Feamster (Georgia Tech) asked how Odin compares to commercial offerings such as those from Meru, Cisco, and Meraki. Julius responded that Odin exposes more low-level hooks to write applications such as mobility managers than the commercial offerings do.

HACK: Hierarchical ACKs for Efficient Wireless Medium Utilization

Lynne Salameh, Astrit Zhushi, Mark Handley, Kyle Jamieson, and Brad Karp, University College London

Awarded Best Paper!

Lynne Salameh discussed how they overcame a limitation with WiFi's medium acquisition overhead and its ramification on TCP's end-to-end throughput. Since every two data packets in TCP require one TCP ACK, this overhead restricts performance as data rates increase, even in the presence of 802.11 frame aggregation and link-layer block ACKs.

The proposed cross-layer solution is named TCP/HACK (Hierarchical ACKnowledgment), which eliminates medium accesses for TCP-ACKs in unidirectional TCP flows by encapsulating TCP-ACKs within WiFi ACK frames. An important design consideration here is that devices using TCP/HACK should coexist with stock 802.11 devices. Furthermore, block ACKs have a hard deadline in that they must be sent within the short interframe spacing duration (SIFS). Given that TCP ACKs may not be ready in time, the block ACKs cannot be delayed since other senders will then acquire the medium. Since TCP ACKs do not have hard deadlines themselves, the TCP ACKs can be appended to the next link layer ACK. Lastly, since a client does not know whether there will be an ACK to send out soon, the access point notifies clients if the access point has any packets destined to them in its transmit queue. This allows clients to not have to guess whether there will be any ACK to be transmitted soon. An implementation of the technique was tested on a software radio platform as well as using simulations with ns-3.

Someone asked whether there are any metrics that worsen when using TCP/HACK. Lynne answered that aggregation makes TCP ACKs bursty anyway, and if you delay TCP ACKs to be encapsulated within the next link layer ACK, you will increase the round-trip time (RTT). Garth Gibson (CMU) asked about the ramifications of breaking modularity, since TCP/HACK is tied to a lower level protocol. Lynne answered that there is always a danger in breaking the layering structure. The final question was on whether there was a performance cost incurred from the

fact that 802.11 ACKs have to be sent at the basic rate of 1 mbps. Lynne responded that 802.11n ACKs are sent at higher data rates.

Best of the Rest III

Summarized by Dimitris Skourtis (skourtis@soe.ucsc.edu)

Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation

Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh Tripunitara, University of Waterloo

The manufacturing of digital integrated circuits (ICs) is being outsourced to multiple teams and external foundries in different locations. Such outsourcing can lead to security threats as soon as any of those foundries inserts a malicious modification to the IC.

The presenter described a threat model where an insider at an external foundry makes a malicious modification. The author then described an example of an attack, where the malicious party attaches a hardware trojan allowing the attacked to be treated as a super user. Next, the presenter gave an example of a modified circuit, where a gate and a trigger were added to allow the malicious party to control when the attack happens. The presenter mentioned that if a malicious modification happens, all IC instances will carry that. Also, whereas with software viruses a patch could be released, this is not true for hardware.

Next, the presenter described how the above problem can be solved by obfuscating the logic of the IC, that is, by hiding certain wires from the view of the third parties. In practice, the IC is split into two or more tiers. The top tier is fabricated in-house and implements the wires that have to remain hidden. The obfuscated circuit is outsourced and is fabricated on other tiers.

The presenter next described a formalization of the level of security provided by circuit obfuscation. In particular, the presenter defined the k-secure gate as one that is indistinguishable from at least k-1 other gates in the circuit. If all gates are k-secure, then the circuit provides k-security. This makes it hard for the attacker to identify a particular gate, because they would have to attack k gates as opposed to a single one.

Next, it was mentioned that it is computationally expensive to find the minimum number of wires that can be hidden while guaranteeing a k-secure circuit. There is a tradeoff between the number of hidden wires and the amount of security. A greedy algorithm was then presented to manage that tradeoff, and was shown to be more effective than randomized selection.

Bill Walker (Fujitsu) asked about detecting malicious modifications. The presenter noted that attackers can disable the attack during testing so that it remains undetected and only be activated afterwards.

Control Flow Integrity for COTS Binaries

Mingwei Zhang and R. Sekar, Stony Brook University

Mingwei first introduced control-flow integrity (CFI), a low-level security property that raises a strong defense against many attacks such as return-oriented programming. Previous work requires compiler support or symbol information to apply CFI. Instead, Mingwei mentioned that their work applies to stripped/COTS binaries, with comparable performance to that of existing implementations.

Mingwei talked about the key challenges: disassembling and instrumenting the binary without breaking the low-level code, and applying their technique to libraries. With respect to disassembling, Mingwei mentioned they are using a mixture of linear and recursive disassembling to mark gaps between pieces of code.

To maintain the correctness of the original executable as well as all the dynamically loaded libraries, they maintain a global translation table (GTT), which gets updated as modules are loaded. Update to GTT is performed by a modified dynamic linker and, in particular, the loader (ld.so).

To evaluate the correctness of their implementation, they successfully applied their method to binaries of over 300 MB (240 MB being libraries). Moreover, Mingwei presented benchmarks (SPEC) to evaluate the runtime overhead (4.29% for C programs), as well as the space (139%) and memory (2.2%) overhead.

Zhiqiang Lin (UT Dallas) asked whether they had encountered any false positives. Mingwei answered that so far they had not, but if there were any, they would discover them.

HotCloud '14: 6th USENIX Workshop on Hot Topics in Cloud Computing

June 17–18, 2014, Philadelphia, PA

Summarized by Li Chen, Mohammed Hassan, Robert Jellinek, Cheng Li, and Hiep Nguyen

Note: The first two sessions of HotCloud '14 were joint sessions with HotStorage '14, and the summary of the keynote can be found in the HotStorage '14 summary on page 91.

Systems and Architecture

Summarized by Li Chen (lchenad@ust.hk)

Academic Cloud Computing Research: Five Pitfalls and Five Opportunities

Adam Barker, Blesson Varghese, Jonathan Stuart Ward, and Ian Sommerville, University of St Andrews

Adam Barker described five pitfalls and five opportunities in academic cloud research in this talk. He argued that academia is pursuing the wrong class of problems and should instead conduct research with higher risk. The core of the problem is the scale of low-level infrastructure that academia has access to is limited, and therefore the research conducted is of lesser value to the cloud computing community.

The first pitfall lies in infrastructure at scale. With more than hundreds of thousands of servers in big cloud computing services, academics can only recreate a small subset of the network,

and cannot replicate the scale. Therefore research may have less value if the researcher does not have access to or partnership with large clouds. The second pitfall is with abstraction, a key feature of cloud computing. Academics see no value in "black box" abstractions, and often need to reimplement the low-level infrastructural components for comparison or prototypes, without support from cloud providers. The third pitfall is with unreproducible results from network simulators, simulations with real-world trace data, and custom evaluation setups. Reproducing the results in papers using these evaluation schemes is nearly impossible. The fourth pitfall is about rebranding cluster/grid computing research as cloud computing. Research on lower levels cannot be tested by academic peers, and research on higher levels may actually have a longer-term effect. The last pitfall is about industrial relations. Current research programs provided by the industry do not address the problem of not being able to access low-level infrastructure.

Researchers should exploit the opportunities in user driven problems. The properties of cloud computing can help solve a number of problems in other domains such as scientific problems. Minimizing cloud resource usage given user derived requirements is also an interesting area. Programming models other than MapReduce should also be investigated, because MapReduce does not fit for all computation tasks. Debugging large-scale applications is very difficult due to the inherent complexity, scale, and high level of abstraction. This area does not receive enough attention from academia. The fourth opportunity lies in Platform-as-a-Service environments. Building environments on multiple cloud infrastructure and providing a high-level interface for users pose interesting challenges. Elasticity is the last opportunity that Adam pointed out. Dynamic provisioning is what differentiates cloud computing from cluster/grid computing.

The first questioner pointed out that there is a huge concern with intellectual property (IP) issues and legal issues. Adam replied that the industry does not feel confident about sharing their facilities because of IP issues. It's really a chicken-and-egg problem: Academia does not have the necessary IP to guarantee deliverables, and industry does not want to share for the same reason. A unified model that resolves these issues in the industrial-academic relationship may be necessary. Another attendee stated that datacenters are drastically different when scaled up, so raising the level of abstraction actually hinders the improvement that can be made by academia. Adam replied that cloud computing and datacenter networking are not the same and require different levels of abstraction. It is important to question all layers in improving datacenter performance, and for cloud computing, a higher abstraction level in fact provides academics with more freedom.

Towards a Leaner Geo-distributed Cloud Infrastructure

Iyswarya Narayanan, The Pennsylvania State University; Aman Kansal, Microsoft Corporation; Anand Sivasubramaniam and Bhuvan Urgaonkar, The Pennsylvania State University; Sriram Govindan, Microsoft Corporation

Aman Kansal started by reviewing the factors affecting the capacity implications of geo-distribution. Latency is the most compelling argument for geo-distribution, as users all over the world would like to be serviced by the nearest datacenters. Another advantage of geo-distribution is failure recovery in case of disasters, but the availability gains come at the cost of excess capacity. Geo-distribution can also exploit regional differences in energy cost.

Aman emphasized the problem of excess capacity, and continued to examine what is the least capacity required. He formulated a linear programming problem with the goal of minimizing the sum of capacities in geo-distributed datacenters. The constraints include latency and capacity to service demand, before and after failures. Aman also identified the trade-off between latency requirement and capacity requirement—tighter latency constraints lead to higher capacity requirements. He showed an interesting result that the excess capacity required by latency and availability jointly is similar to that of latency alone. He also pointed out that routing to the nearest datacenter is not always efficient, especially after a disaster.

Aman went on to describe the open challenges in two aspects: infrastructure and software design. For infrastructure, the previous optimization problem needs to be further examined to consider more factors. Another issue is the fine-grained control of latency and availability for different applications. Lastly, spatial-temporal variations of failures and demands should be exploited to achieve better capacity provisioning.

For software design, Aman emphasized request routing to ensure that the demand is routed to the correct datacenter for efficient capacity provisioning. Placing copies of states and data for efficient user access in geo-distributed infrastructure is another interesting challenge. It is also important for the software to automatically scale the computation with the demand. In the end, Aman came to virtualization, and mentioned that the applications in the cloud should be able to exploit the flexibility of geo-distributed virtualized datacenters.

The first questioner asked: If distribution of clients with different latency classes will affect their formulation, what would be the impact? Aman replied that adding more latency classes can be addressed by small modifications to the formulation, and that they plan on studying the impact in future work. The second questioner asked how their geo-distributed model is affected when the number of servers increases or decreases. Aman answered that depends on the capacity of the infrastructure and how the demand grows over time. In practical cases, land is more important, so the number of servers will not vary significantly. A third person asked about data consistency in distributed data-

centers. Aman replied that their model makes the assumption that the consistency is handled already.

A Way Forward: Enabling Operating System Innovation in the Cloud

Dan Schatzberg, James Cadden, Orran Krieger, and Jonathan Appavoo, Boston University

Dan Schatzberg pointed out that the OSes used in the cloud are often general purpose and not optimized for the cloud. A general purpose OS supports multiple users and applications concurrently, while entire virtual machines in the cloud are often dedicated to a single application. Dan argued for a reduced role for the OS in cloud computing and presented the MultiLibOS model, which enables each application to have its own customized OS.

Dan started by reviewing the unnecessary or relaxed requirements of general purpose OSes in the cloud: support for multiple concurrent users, resource balancing and arbitration, and identical OS (symmetric structure) for all the nodes.

Dan described the MultiLibOS model as combining a general purpose OS with a specialized OS. With this model, applications have flexibility in choosing their own OS's functionalities, from a full legacy OS to a lean customized library. In this way, providing an application with a feature is simple and intuitive, as one need not to go through the labyrinth of a legacy OS. Dan also noted that MultiLibOS makes application and hardware specialization easy, and allows for elasticity and full backward-compatibility.

Dan discussed a few research questions for MultiLibOS. Library development has many known issues, such as configuration, compatibility, "versionitis," fragmentation, reliability, and security. Dan suggested language-level techniques to deal with these issues and noted the importance of efficiently reusing libraries when customizing for different applications; otherwise, a major advantage of MultiLibOS is lost. Lastly, the improvement of a specialized OS needs to justify the cost of development.

The first questioner asked about Dan's intuition of how this was going to work in a virtualized environment. Dan replied that depends on how isolation is implemented in physical hardware; they think it is a good match for virtualized settings. Another attendee wondered whether there's enough headroom to make this work well. Dan replied that their preliminary results show there is a gap, and their design does make improvement to decrease this gap. The final questioner wondered how this is different from RAMCloud. Dan answered that it is along the same line of research, but they focus on giving every application its own customized system.

Software Defining System Devices with the "Banana" Double-Split Driver Model

Dan Williams and Hani Jamjoom, IBM T. J. Watson Research Center; Hakim Weatherspoon, Cornell University

With a botany analogy, Dan Williams showed us that there can be a clean separation of Spike (backend driver) and Corm (hardware driver) in the virtualized cloud, and that the spike and corm do not have to be on the same physical machine.

Dan first identified the incomplete decoupling of system devices in the cloud. The virtual devices are dependent on the physical hardware, which limits the flexibility of the cloud resource management. The split driver model in Xen, while enabling flexibility to multiple access to hardware, fails to provide location independence. To design a generic, software-defined mechanism for device decoupling, he proposed the Banana Double-Split Driver model (Banana for short).

Banana splits the backend driver in Xen into two parts, Corm and Spike. Corm handles multiple accesses to the hardware, and Spike handles the guest OS. Corm and Spike are connected by wire that can be switched in local memory or network connections. Wires are controlled by the Banana controller, which is software-defined and can create on-the-fly reconfigurations.

Dan demonstrated the Banana model by providing an alternative approach to virtualize NICs in Xen, noting that Xen can currently support device-specific complete decoupling of NICs. The management and switching of wires is achieved by integrating the endpoint controller with the hypervisor. Dan mentioned that they augmented the existing Xen live migration mechanism to enable migration of wires and endpoints.

The experimental setup showed the Banana Double-Split model works, but the overhead is large. It is exciting to see that they can live migrate VMs from local cloud to Amazon EC2 without a complex network setup. Dan showed that VM migration is simplified with Banana, but the downtime is increased.

The first questioner pointed out that the guest VM does not have as much detail about the hardware driver. Dan replied that if you want a general framework/API, you will lose some flexibility. The second questioner pointed out that their design requires a taxonomy of all the types of hardware, and wondered whether they had done this work. Dan answered that they had focused on the dependency issues, and their proof-of-concept improves on NICs for now. The final question was about the design's sensitivity to different devices. Dan replied that they had designed something general for all devices. Different devices need to be treated differently, but the authors feel that their model is the way to do it.

Building a Scalable Multimedia Search Engine Using Infiniband

Qi Chen, Peking University; Yisheng Liao, Christopher Mitchell, and Jinyang Li, New York University; Zhen Xiao, Peking University

In this talk, Qi Chen delivered a key insight on how to scale multimedia search in datacenter networks: With low-latency networking, computation time is reduced by using more round-trips to perform searches in a large media collection.

Vertical partitioning is known for its potential scalability for multimedia search engines, yet the large number of indexed

features results in huge communication cost per search query. Therefore it is impractical to implement on Ethernet.

But with high performance networking technologies like Infiniband, vertical partitioning becomes viable, as the round-trip time is only a few microseconds, as opposed to hundreds of microseconds in Ethernet. In this work, they demonstrated the practicality of vertical partitioning by building a distributed image search engine, VertiCut, on Infiniband.

Qi described the two key optimizations in VertiCut. First, VertiCut performs an approximation of k-nearest-neighbor search by stopping early (after getting enough good results). This helps to reduce the number of hash tables read per query. Second, VertiCut keeps a local bitmap at each server to avoid looking up non-existent keys.

For the evaluation, Qi mainly described the comparison with traditional horizontal cutting, dispatch, and aggregate schemes. Qi showed that, with higher network cost (in terms of bytes sent per query), vertical cutting is faster than horizontal cutting on Infiniband. Qi also discussed the effects of the optimizations, concluding that the first optimization results in an 80x speed-up, and the second 25x.

The first questioner asked how the data is stored. Qi answered that their workload is stored in a single DHT table, not on servers. The next questioner noticed a similarity to a previous work on optimal aggregation of middleware and wondered whether they plan to extend their applications. Qi said that in addition to multimedia search, they will have more types of applications in the future. Finally, someone asked whether they have a formal treatment to deal with LSH randomness. Qi said that they have analysis to back up the early stops, and have other approximation methods that they are evaluating.

Mobility and Security

Summarized by Mohammed Hassan (mhassanb@masonlive.gmu.edu)

POMAC: Properly Offloading Mobile Applications to Clouds

Mohammed A. Hassan, George Mason University; Kshitiz Bhattarai, SAP Lab; Qi Wei and Songqing Chen, George Mason University

Mohammed Hassan showed how computation-intensive mobile applications can be offloaded to the cloud more efficiently. He presented a framework that proposed a transparent approach for an existing mobile application to be offloaded. In addition, the authors suggested when the computation should be offloaded and when it should be executed on the mobile device.

Hassan explained that mobile applications are getting more and more resource hungry, but mobile devices are constrained by a limited power supply and resources. Offloading computation to the cloud can mitigate the limitations of the mobile devices. But the current research either requires the applications to be modified or requires a full clone image running on the cloud to be offloaded. Hassan claims that their first contribution is to provide a transparent mechanism for the existing applications to be offloaded without modification. On the other hand, Hassan also emphasized the timing of the offloading decision, which depends on the network bandwidth and latency between the mobile device and the server, and on the server-side load as well. To make the offloading decision more efficiently, Hassan showed that a learning-based classifier would make a more accurate decision for offloading.

Chit-Kwan Lin (UpShift Lab) asked how the bandwidth and latency between the mobile device and the server is measured. Hassan replied that they are monitoring previous values and using a moving average to predict the future bandwidth and latency. He also explained that bandwidth and latency can be well predicted by monitoring the network the mobile is connected to. Michael Kozuch (Intel Labs) suggested that considering remaining battery power in making the offloading decision can help more. Phillip Gibbons (Intel Labs) asked whether the energy consumption is considered here for making the offloading decision. Hassan replied that in future work they are planning to consider the tradeoff between energy consumption and response time for making offloading decisions.

Mobile App Acceleration via Fine-Grain Offloading to the Cloud

Chit-Kwan Lin, UpShift Labs, Inc.; H. T. Kung, Harvard University

Chit-Kwan Lin proposed a novel compression technique that can boost mobile device performance by offloading. Lin included some promising findings about the performance gain of the offloaded performance.

Lin presented the importance of cloud computing for emerging resource-intensive mobile applications. While offloading can augment the computation power of the mobile devices, the bandwidth and latency between the mobile devices and cloud impacts the offloading overhead. With these circumstances, fine-grained offloading may provide more performance gain.

To offload mobile computation, it is necessary to have a replication of the application in the cloud side to execute the offloaded application there, and to synchronize the server-side replication's memory blocks. Lin showed a novel technique to minimize the synchronization data transfer overhead by compression. In short, the change in the mobile device's state is compressed and sent to the server side. The server side synchronizes by uncompressing the changes and thus updating itself. In this way, offloading can be done without object marshaling and with less overhead. At the end, the presenter demonstrated the effectiveness with a handwriting recognition application.

Someone asked how change is sent to the server. Lin responded that the changes are sent continuously. Another person asked how the server side was executing the offloaded application. Lin said that same exact application was running off the server side. Ymir Vigfusson asked about overhead if there are lots of writes and the mobile device state changes a lot. Lin replied that in

that case the compression technique might not help that much because there would be a lot of overhead.

Leveraging Virtual Machine Introspection for Hot-Hardening of Arbitrary Cloud-User Applications

Sebastian Biedermann and Stefan Katzenbeisser, Technische Universität Darmstadt; Jakub Szefer, Yale University

Sebastian Biedermann proposed an architecture to improve security settings of network applications in a cloud computing environment. Biedermann proposed a technique to locate and access memory locations of another VM for runtime analysis of applications on VM.

Biedermann introduced his presentation by citing related work, "hot-patching," which enables runtime analysis of another VM from the virtual machine introspection (VMI) by accessing the VM's memory. Hot-hardening is a similar approach that continuously and transparently improves the security-related configuration of running apps in a VM. At first the security or configuration setting (it may be a file in the memory or storage) of the target VM is identified and located. Then the VM is cloned for inspection. After the VM is cloned, the settings of the cloned VM are replaced or written with a different configuration to see its impact on applications. Finding an application's settings file in the VM's memory or storage is challenging. Biedermann showed that the setting file can be found by searching for certain settings' patterns in the VM's memory. Biedermann finally showed the framework's effectiveness with some real-world applications (e.g., MySQL and OpenSSH).

The first questioner asked how the configuration files were detected. Biedermann answered that they were using some heuristics to look for the setting's pattern. The second questioner wondered about the latency of VM cloning. Biedermann replied that it takes only few seconds to clone, which is acceptable. The last questioner wanted to know how the settings changes are injected in the cloned VM. Biedermann replied that it was done by changing the memory/page of the target VM.

Practical Confidentiality Preserving Big Data Analysis

Julian James Stephen, Savvas Savvides, Russell Seidel, and Patrick Eugster, Purdue University

Julian James Stephen proposed a framework to encrypt data for MapReduce work in the cloud. Security and data confidentiality are big concerns for cloud computing, where users have to trust third-party cloud providers with private data. The proposed framework showed that computation can be conducted in the cloud over the encrypted data while the server side is not aware of the actual content.

Stephen started his presentation by stating that the cloud has a big potential for carrying computation, but it also comes with the potential for security breaches like a data leak. But data can be encrypted with fully homomorphic encryption (FHE) so that the server side may carry the computation without knowing the actual content. But FHE is associated with high overhead, while partial homomorphic encryption (PHE) can keep the encryption overhead acceptable and is capable of performing certain operations. Stephen proposed a framework, Crypsis, that transfers Pig Latin script for MapReduce to accept encrypted data for computation. With an example, he demonstrated how a simple Pig Latin script can be transferred to work with encrypted data. Here the data is encrypted by a different encryption technique, and the original operations are transformed to operate on encrypted data by a user defined file (UDF). Stephen then compared the time to execute original and encrypted scripts and observed a three times overhead for the encrypted operations. The presentation also included some limitations: namely, the proposed framework does not support iterations; the UDF has to be defined by the user; and although the data is encrypted, the data access pattern is exposed when computation is carried in the cloud.

In the question and answer session, Phillip Gibbons (Intel Labs) asked how encrypted data is read on the client side. Stephen answered that the client side decrypts the data to find the result.

Keynote Address

Summarized by Cheng Li (chengli@mpi-sws.org)

Programming Cloud Infrastructure

Albert Greenberg, Director of Development, Microsoft Azure Networking

Albert Greenberg presented the framework they built inside Microsoft to allow developers to easily manage the large-scale cloud system. He started his talk by showing that the cloud system has grown very fast in Microsoft. In the past four years, computation and storage have doubled every six months, and a significant number of customers have signed up to use the Azure services. In addition, applications are not running in separated environments; instead, they are concurrently sharing resources within a single datacenter or across multiple datacenters.

All these trends urgently require an efficient and easy-to-use management application, which should provide an unambiguous language for architects to describe intent, codify design, and generate full details of the design, and allow different applications to consume data. To achieve this goal, Albert's team proposed NetGraph, an abstract graph model of network, to specify arbitrary network topology and state in an XML-like fashion. To be more specific, he showed a few adaptations of the graph model: physical network graph, data plane graph, control plane graph, and even the overlayed network graph.

Without the graph model, in the conventional buildout scenarios, a group of engineers played a very important role in transforming the high-level design into a deployed and configured system. PDFs and spreadsheets that described the design and are often in vendor-specific formats were exchanged among them to justify and debug the design. The obvious drawback of this approach is that it is really impossible to automate. To ease the developers' work and make the design highly dependable, they built a network graph generator and a network graph service to make the best use of the graph model. Developers could use

the reusable and extensible plugin modules in the generator to enforce the design principles. Additionally, the generator could produce both human-readable and machine-readable description files. The graph service stores the detailed design information in an in-memory graph database, and offers APIs for fetching and updating the information.

In addition to the automation, Greenberg mentioned the problems they found while managing the large-scale systems, such as unexpected states, interference, dependencies, and so on. To resolve these problems, they chose a state driven approach, where they could model dynamic network changes in a network state service (NSS). NSS knows all target states (good states) and all observed states (maybe bad), and periodically checks the difference between these two types of states to figure out and reconcile unexpected behaviors.

Following the talk, many interesting questions arose. The first was about the purpose of maintaining versioned objects. Greenberg replied that all objects in the graph database are versioned since they have to be able to roll back. The second question concerned the quiescent point before applying updates. Greenberg said that it is not practical to identify the quiescent point. Instead, they could roll back if any mistakes had been made. They also try to make the updates fast and incremental. The last question was about access control (ACL). Albert pointed out that ACL can conflict and overlap, so they designed a few automated methods to constantly check ACL (e.g., set comparison) and flag conflicts for an admin to investigate/resolve.

Diagnostics and Testing

Summarized by Hiep Nguyen (hcnguye3@ncsu.edu)

A Novel Technique for Long-Term Anomaly Detection in the Cloud

Owen Vallis, Jordan Hochenbaum, and Arun Kejariwal, Twitter Inc.

Owen began by noting that existing work in anomaly detection does not work well when dealing with long-term anomalies such that just using the median is not good due to pronounced trend. Owen described the observation with Twitter production data that the underlying trend often becomes prominent when they look for a longer time span (i.e., more than two weeks) using time series data.

Owen then described the experience with exploring two approaches to extract the trend component of a long-term time series using STL Trend (seasonal, trend, and irregular components using Loess) and Quantile Regression. Neither of these two worked well in their experiments. He then introduced a technique called Piecewise Median to fix the limitations of these approaches. This technique computes the trend as a piecewise combination of short-term medians.

Someone asked whether this technique is used in the real production system and how the author would apply it. Owen said they tested the technique with the production data, and they are working with the team to deploy it in a real production system. Another questioner asked whether the authors consider the medians of nearby windows. Owen said it would be definitely helpful to consider those medians.

PerfCompass: Toward Runtime Performance Anomaly Fault Localization for Infrastructure-as-a-Service Clouds

Daniel J. Dean, Hiep Nguyen, Peipei Wang, and Xiaohui Gu, North Carolina State University

Daniel started with describing the common problem with multi-tenant cloud systems where the observed problem may come from external sources such as resource contention or interference or because of the software itself. If the admin can determine whether the performance anomaly is from an external fault, system administrators can simply migrate the VM to another physical node to fix the problem.

Daniel then introduced a system named PerfCompass that uses a system call trace analysis technique to identify whether the root cause of a performance anomaly is an external fault or is an internal fault. The main idea of the technique is based on the observation that the external fault will have a global effect on the application, meaning most of the threads will be affected. On the other hand, if the performance anomaly is caused by an internal fault, only a subset of the threads is affected. Finally, he described the results on testing the system with a set of real internal faults and typical external faults, showing that the system performed well with those tested faults.

John Arrasjid (VMware) asked whether the authors consider the arguments of system calls. Daniel said that it would be definitely helpful to consider that. He also mentioned that the way the system does segmentation helps in grouping system calls with similar arguments. Someone from VMware commented that the authors may want to look at the console log because it might be difficult to enable system call tracing in the real production systems. Daniel said that not all applications generate a console log, and system call tracing is lightweight.

MrLazy: Lazy Runtime Label Propagation for MapReduce

Sherif Akoush, Lucian Carata, Ripduman Sohan, and Andy Hopper, University of Cambridge

Sherif described MrLazy, a system that relies on lineage (i.e., origin information for a piece of data) to ensure that potentially sensitive data is checked against sharing policies applied to the data when it is propagated in the cloud. The motivation for the work is that existing work has various deployment challenges and runtime overhead issues. Sherif stated that checking data within a given trust domain continuously is not necessary and is the main source of the overhead. MrLazy delays the enforcement of data dissemination policies to the point where data crosses a trust boundary.

Sherif then described the results that the authors performed on a MapReduce framework. The results showed that MrLazy can significantly improve the job running time.

There were no questions.

Mechanisms and Architectures for Tail-Tolerant System Operations in Cloud

Qinghua Lu, China University of Petroleum and NICTA; Liming Zhu, Xiwei Xu, and Len Bass, NICTA; Shanshan Li, Weishan Zhang, and Ning Wang, China University of Petroleum

No presentation, paper only.

The Case for System Testing with Swift Hierarchical VM Fork

Junji Zhi, Sahil Suneja, and Eyal de Lara, University of Toronto

Junji started by stating that software testing is challenging because there are a lot of test cases that need to be executed, which may take a very long time if performed sequentially. He then gave an example of testing MySQL software. Junji then motivated his work with the observation that multiple steps during testing are shared, the test cases share the same code base, and a lot of test cases need to reuse the state of another test case; the authors concluded that if they could reuse the state of test cases, they could speed up the testing process.

Junji then described the idea of using VM fork to clone the VM that has the state of the finished test case available to use for multiple other test cases, thus allowing reuse and parallel testing. He went on to describe how this would improve the testing time in the MySQL example.

Someone asked whether the authors needed to assume that test cases are deterministic and whether they had any thoughts on applicability to non-deterministic test cases. Junji said that they need to assume test cases are deterministic. Someone else asked whether they can use OS fork instead of VM fork. Junji replied that running multiple processes might end up not working because of resource-sharing.

Economics

Summarized by Robert Jellinek (jellinek@cs.wisc.edu)

BitBill: Scalable, Robust, Verifiable Peer-to-Peer Billing for Cloud Computing

Li Chen and Kai Chen, Hong Kong University of Science and Technology

Li Chen presented work on BitBill, a system that ensures verifiable accounting of billable events in the cloud. He noted that more companies are using cloud computing, but that verifiable billing is still an issue for both providers and tenants.

In particular, providers may have trouble accounting precisely for all resource usage, which may in fact be detrimental to them since they can undercharge. Tenants, on the other hand, cannot perform an audit to verify that they are being billed correctly, because the actual physical resource consumption is behind a layer of abstraction. Tenants cannot trust providers under the current model.

Chen said that the lack of trust is currently impeding a wider adoption of cloud computing, and he presented several trust models: two existing models, and the authors' proposed model. The first model is that of unconditional trust, which is currently used in practice by commercial cloud providers. In this model, the tenant trusts the provider to accurately record tenants' resource usage and to bill accordingly. The tenants have no way to verify that they are billed accurately. The second model—the third-party trust model—uses a trusted third party to verify that resource accounting and billing are accurate. One problem with this model is that it introduces a central point of failure: the third party. Furthermore, the third party must itself have enough resources to perform accurate resource accounting, which could turn out to be a bottleneck.

Chen then introduced a third model, the authors' public trust model, where trust is distributed across all nodes in a network. The nodes maintain a single global history of billable events, which the authors implemented using a peer-to-peer (p2p) network to maintain resource accounting information across all participating nodes. Here, the only assumption is that the majority of nodes in the network are honest (i.e., will not introduce false events to the global log, or omit true ones). This is reinforced by the fact that all nodes share the same physical resource pool, and so one primitive resource, such as a CPU cycle on a single core, cannot be billed to two tenants.

The authors' implementation of this public trust model uses the Bitcoin-like solution to the Byzantine Generals Problem to ensure they have a trustworthy distributed log of billable events, even in the presence of untrustworthy individual nodes. Here, every billable event is broadcast to all nodes and is signed by the announcing party. To avoid false announcements, they use a simpler version of the proof-of-work (PoW) technique used in Bitcoin, where any announcing party must solve a computationally intensive problem to send along with the announcement. These PoW problems are NP-hard, so that they are easy for nodes to verify but hard for them to forge, ensuring that double billing announcements do not occur.

Chen then briefly explained the implementation of BitBill, which uses a Merkle tree so that every non-leaf node is labeled with the hash of the labels of its children nodes. Once a node finishes the PoW problem, it broadcasts its block to all other nodes, which then verify that block and use it to construct the next block. This yields the important property that the existence of an item in the log means that a network node has accepted it, and the blocks subsequently added to the log further affirm its validity. Ties are broken such that a given node works on the longest chain it sees, and nodes add any blocks they've missed by pulling them from future announcements they receive.

Chen noted that in their evaluation so far, BitBill appears to be much more scalable than the third-party-verifier model, and they are continuing evaluation. He then discussed deployment, resource monitoring, and security, saying that BitBill can be distributed by providers for users to install as a package or included in the user's VM, that BitBill can be used as the basis to extend existing work on verifiable resource accounting, and that due to the PoW approach, BitBill is secure as long as the majority of participating nodes are honest.

Michael Kozuch (Intel Labs) asked how often verification needs to happen, and what a standard policy would look like. Chen answered that it would depend on how the provider would want to charge the tenants, and that sampling and verification could happen at varying granularities.

A Day Late and a Dollar Short: The Case for Research on Cloud Billing Systems

Robert Jellinek, Yan Zhai, Thomas Ristenpart, and Michael Swift, University of Wisconsin-Madison

Robert Jellinek presented work on cloud billing systems, focusing on existing systems' lack of transparency, long update delays, unpredictability, and lack of APIs.

Jellinek began by noting that despite the fact that much attention has been paid to performance, reliability, and cost studies of the cloud, there has been no study of the billing systems themselves. The predominant pay-as-you-go pricing model relies upon complex, large-scale resource-accounting and billing systems that are not fully understood by cloud computing consumers.

The main question the authors considered was how one can track resource usage in real time and at fine granularity while maintaining accuracy and not hurting performance. They investigated Amazon Web Services (AWS), Google Compute Engine (GCE), and Rackspace public cloud. Jellinek noted that they were able to reverse-engineer the timestamps corresponding to various billing events, uncover several bugs in the providers' billing systems, detect systematic undercharging due to aggregation or caching, and characterize the performance of billing latency, which turned out to be substantial across all platforms.

Jellinek then described the methodology for their measurement study, which involved instrumenting providers' API calls to collect timestamps of all important instance lifetime events, largely by polling the APIs for instances' state. They would then launch an instance, execute a workload to test compute-time billing thresholds, storage, or network usage, fetch instance-local data related to the workload in question, terminate the instance, and then poll providers' various billing interfaces to check for updates. Billing latency, which they define as the time between when a resource is consumed and when the corresponding charge becomes available on a given billing interface, is recorded when a billing update is registered.

Jellinek then described various billing interfaces, including the Web-based GUI interfaces available for all three providers, and the additional CSV interfaces and Cloudwatch monitoring service available on AWS. Collecting information from the GUI interfaces required screen scraping, and none of the interfaces were particularly user-friendly. No providers offered billing APIs.

The authors found that billing updates would not necessarily occur atomically and that they occurred with high and unpredictable latency. Among other things, this made experiments difficult, since it was necessary to wait for longer than the greatest observed latency to be sure all updates had been registered. AWS, GCE, and Rackspace updated with average latencies of 6 hours 41 minutes, 22.5 hours, and 2.2 days, respectively, and with high variance. This shows that billing updates are both slow and unpredictable, which he claimed is bad for consumers who wish to optimize their deployment decisions.

Jellinek then described their experiments to measure when billing for an instance begins and ends, noting that this is ambiguous since most providers are not specific enough in their documentation or in the timestamps they provide. This means that, if a user thinks she has only run an EC2 instance for 3590 seconds, she may in fact get charged for two hours of usage, depending on how she measures an instance hour. The authors found that, despite the fact that they were able to determine what timestamps correspond to the start and end of billing for the three providers, they were not able to measure this precisely. This is due to the semantic gap between the providers' knowledge of their billing timestamps and the customers' knowledge. If the provider does not report its record of the relevant timestamps, a customer cannot know them precisely since they have to poll the provider's APIs for updated instance-state information. This is subject to jitter from variable network latency, server response time, and polling granularity. He then described a bug they found in EC2 that would yield two minutes on average of free compute time under certain conditions relating to when the instance was terminated.

In the rest of the talk, Jellinek described results on storage and network tests. The authors found a bug in Rackspace persistent storage volumes that led to overcharges when volumes became stuck in an intermediate stage, unusable but still being billed. He then noted that the authors found that billing for IOPS in EC2 was subject to a substantial amount of aggregation on sequential reads and writes, which leads to underbilling for the customer. While this may seem good, the downside is that billing for IOPS in EC2 is still opaque and ultimately unpredictable to the customer. Finally, he noted the authors' discovery that billing for networking is also systematically slightly undercharged in EC2, and that they discovered a bug in Rackspace's network billing that led to more severe but less common undercharges.

In concluding, Jellinek suggested that providers should offer a billing API in which they expose key parts of their internal billing-related data and metadata (billing start/stop timestamps, network and storage billing data, etc.). He closed by noting that important future work could be done to better understand the tradeoffs inherent in implementing transparent, real-time billing interfaces, and how we could optimize billing interfaces, and the underlying resource-accounting mechanisms, in light of these tradeoffs.

An attendee from IBM asked whether they had tried testing from different locations other than from the university. Jellinek responded that they had not, but that that was definitely a good idea to pursue in verifying the results.

A second attendee asked whether they had tried creating a small cloud environment and polling it to see whether the actual resource consumption by the guest OS matched the cloud environment's measurements. Jellinek responded that they had not done that, but that it sounded like another good idea to pursue to verify their results.

A representative from VMware asked whether the authors had considered viewing billing as a statistical process, rather than as a process of exact resource accounting. Jellinek responded that, from the conversations he's had, it seemed that cloud providers are aware that exact resource accounting is a hard engineering problem that requires a significant amount of engineering effort and hardware. In practice, it is definitely conceivable that behind the scenes there is a certain amount of sampling and rounding down, such that any inconsistencies are in favor of the consumer, but ultimately allow the provider to conserve costs associated with exact resource accounting. This is speculation though, since to really know, one would have to understand the underlying mechanisms, which are proprietary.

A Case for Virtualizing the Electric Utility in Cloud Datacenters

Cheng Wang, Bhuvan Urgaonkar, George Kesidis, Uday V. Shanbhag, and Qian Wang, The Pennsylvania State University

Cheng Wang presented work on virtualizing the electric utility in cloud datacenters. He began by discussing how expensive it is to power a datacenter; the cost of building the IT infrastructure is often comparable to building the power infrastructure that is needed to keep the servers powered. The same is true for the utility bill that is used to power the servers each month. These are both on the same order as the IT investment itself.

Wang then discussed how a datacenter currently recoups operating expenses from tenants, and how it should actually be done. Today, operating expenses are recouped by charging for virtualized IT resources such as compute time, storage, and network resources. However, electricity is billed in a very different way. One common way it's billed is "peak-based pricing," which differs from how we consume electricity at home. Home consumers spend a certain amount per kilowatt-hour (kWh) of electricity consumed, and that's it. But for large consumers such as datacenters, they pay this charge as well as an additional charge that is connected with their pattern of consumption. Essentially, they pay more if their consumption is more bursty. So they may pay $0.05/kWh for usage up to some point, but would then pay $12/kWh for peak power consumption drawn above some wattage at a given point in time. The takeaway, says Wang, is that there is a peak-to-average pricing ratio of 3:1, and this ratio affects the economics of cloud computing. The question is how this gets passed on to the consumer.

Wang claims that it is passed on to consumers unfairly, in a way that does not accurately reflect cloud consumers' share of the peak-power consumption costs incurred by the cloud provider. In particular, he noted two shortcomings: a lack of fairness in how tenants get charged and a loss of cost-efficacy for both cloud tenants and providers.

To understand the unfairness, Wang encouraged the audience to consider two tenants that consume the same amount, but where tenant T1 has low variance, and T2 has extremely high variance, including consumption at peak times. In the current model, both tenants are charged the same amount because they pay fixed prices for virtualized compute resources, but T2 imposes a higher cost on the cloud provider than T1, because T2 contributes to peak-power demand that is three times more expensive than non-peak power.

The solution, Wang claims, is to virtualize the utility so that the energy costs a tenant incurs are passed on to them and not redistributed unfairly across all tenants. In essence, this means passing on the pricing structure of electricity to tenants themselves, so that these prices reflect the value the tenants derive from using that power. Wang related this to building exokernels and letting applications carry out their own resource-management solutions. Here, with a virtualized utility, tenants will be incentivized to use their resources more efficiently and will manage their usage more carefully based on those new incentives.

In practice, Wang says that this approach should be used with large, long-lasting tenants. It will be more difficult for them to take this extra factor into account and to optimize for cost, but will ultimately let them feel like they are really operating within the datacenter, with all its associated concerns, and provide a more equitable distribution of costs.

Phillip Gibbons (Intel Labs) noted that the main challenge seemed to lie in the peak pricing model itself. Passing on prices according to that structure means that you never want to be the customer who contributes to peak power, but you want to be right after them. Gibbons said that this seems like an artificial artifact of that pricing model. Wang responded that peak power is just determined by the behavior of the consumer, not time of day or anything else. Gibbons responded that it's so easy to game the system then, by just avoiding contributing to peak power consumption.

An attendee from Boston University noted that Wang had made a comparison to exokernels, and that one of the main challenges exokernels faced was that of aggregation: When you lower the level of abstraction, it makes it harder to perform aggregation. He asked whether cloud computing would similarly lose out on the benefits of aggregation if this layer of abstraction is removed. Wang replied that he did not suggest that the existing interface should be replaced but, rather, augmented. In his proposed interface, tenants would access their normal interface but also see metrics about how much they're contributing to peak power consumption.

HotStorage '14: 6th USENIX Workshop on Hot Topics in Storage and File Systems

June 17–18, 2014, Philadelphia, PA

Summarized by Rik Farrow, Min Fu, Cheng Li, Zhichao Li, and Prakash Narayanamoorthy

Keynote Address

Summarized by Rik Farrow (rik@usenix.org)

The Berkeley Data Analytics Stack, Present and Future

Michael Franklin, Thomas M. Seibel Professor of Computer Science, University of California, Berkeley

Franklin began by explaining that, in the Algorithms, Machines, and People Lab (AMPLab), they have been building the Berkeley Data Analytics Stack (BDAS), pronounced Bad Ass. BDAS is composed of many elements that were introduced at past HotCloud workshops, and Franklin told us he would walk us through the stack, what it is and why they built it. The reason for BDAS is that there are cascades of data being generated: logs, user generated, scientific computing, and machine to machine (M2M) communication. Instead of defining big data, Franklin provided the example of Carat, an application that collects data on apps and power use on smartphones, sends it to be processed using AWS and a BDAS framework called Spark, and then provides personal recommendations to the users of the apps about any energy hogs they may be running. What's interesting about big data is that you can see things that you can't with less data.

In order to make a decision, you have the envelope of time, money, and answer quality. You want to stay within that envelope, and the first thing researchers and programmers do is increase performance. When they hit the wall, they can trade off for less quality, or pay more for better quality. Another way to think about this is via algorithms, machines (warehouse computing), and people. In AMPLab, they want to use these resources to solve the big data problem.

MapReduce is a batch processing algorithm that proceeds through grouping and analysis, but there are a lot of other things that people do with databases. MapReduce can be specialized for streaming, working with graphs, or targeted for some other design point. The BDAS approach is to generalize, rather than specialize, MapReduce by adding general task DAGs (directed acyclic graphs) and data sharing, making streaming, SQL, and machine learning not just possible but faster than the specialized versions of Hadoop MapReduce. Spark, the BDAS core execution engine, is smaller than Hadoop, Storm (stream processing), Impala (SQL), Giraph (Graph), and Mahout (ML). And even with other modules added to handle machine learning, graph processing SQL, and streaming, Spark is still smaller than any of the other popular tools that can do just one of these activities. Like these other tools, Spark is open source, which meant, among other things, that students had to decide to produce quality code instead of producing more papers.

Franklin went on to describe several other projects, starting with MESOS, a system that allows sharing a cluster with different frameworks, like Hadoop, Storm, and Spark. Tachyon is an in-memory, fault-tolerant storage system that can be shared across different frameworks. Spark is now Apache Spark, and Hadoop may fade away, replaced by Spark or something else, not bad for a student project (Matei Zaharia's, who wrote about his creation for *;login:*).

RDD (Resilient Distributed Datasets), a key part of Spark, came out of a desire to improve the performance of Hadoop for machine learning. RDD caches results in memory rather than on disk, as Hadoop does, taking disk processing out of the critical path. RDDs maintain fault tolerance by including the transformations needed to recreate immutable stores of data. RDD also works well for SQL (Shark), which allows Hive queries to run without modification 10x to 100x faster. SparkSQL is inside of Spark 1.0, and Shark will be ported to run within Spark. BlinkDB provides a SQL interface that provides approximate answers, the benefit being speed by using sampling and displaying the error range. Future work will add the ability to perform online transaction processing (OLTP), which will require modifications to the way that RDD works to support frequent, concurrent updates. Graph processing (GraphX) is another ongoing project.

Franklin ended his talk with reflections and trends. While "Big Data" has the word "Big" in it, the real breakthrough isn't scalability—it's really about flexibility. With a traditional database, you begin a process called ETL (extract, transform, load), and import the data in a vault where you get a promise that your data will be reliably stored. In this type of database, there is one way in and one way out, but the price you pay is you lose access to that data except via SQL. In Hadoop, you split that up into storage and multiple methods of accessing that data. Another type of flexibility is that there is no schema: data can be unstructured. It can be structured (SQL schema), semi-structured (XML), and unstructured (Hadoop and others).

Also, in big data, people have ignored single node performance. That needs to change, because for small clusters, a single node is more efficient: Distributed systems are hard. In the AMPLab, they want to make BDAS work better for uses that require random write and random read, neither of which Spark and RDD are good at. These are the directions AMPLab is going.

Franklin finished a bit after his allotted time, and so Q&A was limited to a single question. Steve Muir (Comcast) pointed out that Franklin didn't talk about programming languages or traditional systems stuff, and wondered whether that has been a difficult change. While the Enterprise has adopted Java, Spark was written in Scala. Are there benefits from abandoning C++? Franklin replied that Steve is right, particularly with single node performance. Where you need to pay attention to low-level stuff is when you start benchmarking. Cloudera Impala (SQL) is written in C++. But there are things you can do to avoid JVM issues.

Money, Batteries, and Shingles

Summarized by Min Fu (fumin@hust.edu.cn)

qNVRAM: quasi Non-Volatile RAM for Low Overhead Persistency Enforcement in Smartphones

Hao Luo, Lei Tian and Hong Jiang, University of Nebraska, Lincoln

Hao Luo argued that since smartphones equipped with irremovable batteries have become more popular, it is time to rethink the memory volatility in smartphones. Luo proposed qNVRAM to reduce performance overhead without decreasing persistency level to less than traditional journaling and double-write persistency mechanisms.

Luo first introduced existing persistence enforcement mechanisms (including journaling and double-write schemes) in smartphones, which result in significant overhead due to additional I/Os. Luo then introduced four failure modes in Android smartphones, including application crashes, application hangs, self-reboots, and system freezes. All four modes could result in loss of application data. Given that more and more smartphones are equipped with irremovable batteries, the DRAM can be considered as a quasi NVRAM. Luo then proposed qNVRAM, an easy-to-use memory allocator. When one of the four failure modes happens, the application data in the qNVRAM pool can be restored. qNVRAM significantly speeds up the insert, update, and delete transactions in the SQLite use case.

Someone asked how to ensure data integrity in physical memory. Luo answered that ECC is implemented in the kernel and checksums are used in the database. Xiaosong Ma (Qatar Computing Research Institute) asked about the energy consumption. Luo answered that qNVRAM can reduce energy consumption. Someone asked, what if I dropped my phone on the floor? Luo answered that this rarely occurs. Dai Qin (University of Toronto) asked whether the data would be lost if the battery has died. Hao's answer was no.

Novel Address Mappings for Shingled Write Disks

Weiping He and David H.C. Du, University of Minnesota, Twin Cities

Weiping He proposed several novel static logical block address to physical block address mapping schemes for in-place update Shingled Write Disks (SWD). By appropriately changing the order of space allocation, the new mapping schemes improve the write amplification overhead significantly.

He started by describing SWD. He explained that in-place SWD requires no garbage collection and complicated mapping tables of out-of-space SWD, but suffers from the write amplification problem. He observed that a simple modification of the writing order of the tracks can reduce the write amplification, such as writing tracks 1 and 4 first. He then presented three novel mapping schemes, including R(4123), 124R(3), and 14R(23). These mapping schemes could improve update performance significantly when SWD space usage is less than 75%.

The first question was whether there are any workloads that revert the advantage of the new address mapping schemes. He replied that general workloads won't revert the advantage. The second question was whether the new address mapping schemes are designed to take advantage of temporal localities. He's answer was no. Nitin Agrawal (NEC Lab) asked about the age of the disk model used in the experiments. He replied that it's about 10 years old but is the newest they can get. Lots of researchers are still using it.

On the Importance of Evaluating Storage Systems' $Costs

Zhichao Li, Amanpreet Mukker, and Erez Zadok, Stony Brook University

Zhichao Li argued that evaluating storage systems from a monetary cost perspective becomes increasingly important. Li built a cost model, and evaluated both tiering and caching hybrid storage systems.

Li started by describing two kinds of hybrid storage systems: tiering and caching architectures. Li said performance alone is not enough to evaluate a hybrid system and dollar cost matters. An empirical TCO (total cost of ownership) study is also lacking when systems deploy SSD. Li then presented a cost model for hybrid systems, including upfront purchase as well as TCO. Li compared the two architectures of hybrid storage systems in terms of monetary cost. The results are workload-dependent. Li also said the cost model has several limitations, such as not including computer hardware, air-conditioning, and so on.

Three people asked questions about the cost model, including someone from Red Hat, Xiaosong Ma (Qatar Computing Research Institute), and Peter Desnoyers (Northeastern University).

A Brave New World (of Storage System Design)

Summarized by Zhichao Li (lzcmichael@gmail.com)

Towards High-Performance Application-Level Storage Management

Simon Peter, Jialin Li, Doug Woos, Irene Zhang, Dan R. K. Ports, Thomas Anderson, Arvind Krishnamurthy, and Mark Zbikowski,University of Washington

Simon Peter proposed a novel architecture to move the operating system storage stack off the data path for optimized performance. The idea is based on the observation that the operating system storage stack is becoming the bottleneck in the I/O path.

Simon began the presentation by stating that file system code is expensive to run. He illustrated the transition from today's storage stack to their storage architecture where the storage stack (block management and cache) is moved to user-level. Simon then discussed the proposed architecture in more detail. In their storage hardware model, the kernel manages virtual storage devices and virtual storage areas (VSA). The VSA maps from virtual storage extents to physical storage extents, and it is guaranteed that there is at least one VSA per application. The VSA also handles the global file name resolution and uses persistent data structures for high-level APIs.

They implemented a case study system using FUSE based on the idea illustrated above. Evaluation against Redis showed

that their system cut SET latency by 81%, from 163 µs to 31 µs. Simon then summarized their study by stating that leveraging application-level storage eliminates I/O bottleneck and achieves a 9x speedup compared with Redis, and it scales with CPUs and storage hardware performance.

Someone from VMware asked for the statistics of context switches in the system. Simon replied that there was basically no context switch since only library calls were involved. The attendee then asked whether it is possible to just modify OS for the same purpose. Simon said no and stated that doing so increases the attack space within the kernel and will only make the complex system even more complex. Geoff Kuenning (Harvey Mudd College) asked how the system scales when there are millions of files being accessed. Simon replied that it is possible that some applications will slow down, but other applications can access millions of files efficiently. Peter Desnoyers (Northeastern University) asked what the authors think of customizing OS functionality for different applications either in kernel or in user-level. Simon replied that it is hard to answer and continued by stating that user space is easy to experiment with and working with the kernel is complex, and so they choose to go with user-level. Steve Swanson (UCSD) asked about the fundamental difference between file access and block access. Simon replied that this is a good question and files have names associated with them. Margo Seltzer (Harvard School of Engineering and Applied Science and Oracle) asked whether Simon could comment on Exokernel since Exokernel appears similar to Arrakis. Simon replied that the difference lies in the fact that hardware is now different, which matters more for storage.

NVMKV: A Scalable and Lightweight Flash Aware Key-Value Store

Leonardo Mármol, Florida International University; Swaminathan Sundararaman and Nisha Talagala, FusionIO; Raju Rangaswami, Florida International University; Sushma Devendrappa, Bharath Ramsundar, and Sriram Ganesan, FusionIO

The idea that Leonardo Mármol presented for a flash aware key-value store is to examine the Flash Translation Layer (FTL), instead of the upper-level key-value software, to leverage SSD in an optimal way.

Leonardo began by introducing key-value stores and then discussing the limitations of existing solutions—for example, better performance only on HDD and older SSDs, requiring compaction/garbage collection and introducing a write amplification problem, from 2.5x to 43x in one example. Leonardo then took a look at FTL, which manages data in a way similar to a key-value system, and proposed to move almost everything (except the key-value hashing mechanism) to the FTL for optimal efficiency. This is a new approach by cooperative design with FTL to minimize auxiliary write amplification, maximize application level performance, and leverage FTL for atomicity and durability by extending the interface. Leonardo then discussed the classes of key-value store: disk optimized and SSD optimized.

Leonardo next talked about the design: Sparse address mapping (LBA = hash(key)) leads to FTL sparse mapping, and translates logical to physical addresses. This is made possible by the extended FTL interface (i.e., atomic write and atomic trim; iterate, query an address). Leonardo further stated that hashing and collision is achieved by polynomial probing: Their software tries eight positions before failing. In their evaluation, microbenchmark results are generally positive and beat LevelDB even at low thread counts and without FS cache; the YCSB benchmark shows that their system beats LevelDB in all conditions as well. Leonardo concluded by proposing FTL cooperative design for simple key-value store design and implementation for high performance and constant amounts of metadata.

Michael Condit (Red Hat) asked why LBA and PBA are two to three times larger in space. Leonardo replied that it is because of a more efficient caching implementation. Margo Seltzer asked why they only compared with LevelDB when there are lots of other available key-value stores. Leonardo replied that there is no particular reason why they chose LevelDB and it is future work to compare against other key-value stores. Margo also commented that there is paper from FAST '14 that looks into the cooperative file system design with SSD, and suggested Leonardo look into that. Leonardo agreed. One attendee from VMware asked about operations for key lookup. Leonardo replied that it depends on the hashing and the key being looked up. In most cases, Leonardo continued, there is only one I/O for key lookup, and under other cases, it may need multiple operations. They have set a limit of eight lookups before giving up.

FlashQueryFile: Flash-Optimized Layout and Algorithms for Interactive Ad Hoc SQL on Big Data

Rini T. Kaushik, IBM Research—Almaden

Rini Kaushik introduced FlashQueryFile: a flash-optimized layout and algorithm for interactive ad hoc SQL queries on big data. The idea is to optimize the data format in consideration of the underlying SSD characteristic for optimized big data analysis usage of flash.

Rini started the talk with the motivation that there are many use cases for ad hoc SQL queries, and storage plays an important role in ad hoc big data queries. Flash in a big data stack is faster and cheaper than DRAM, is non-volatile, and incurs lower energy and better total cost of ownership. Rini then discussed the challenges in flash adoption: Systems are currently HDD optimized; suboptimal performance/dollar on flash; flash sequential bandwidth is only 2–7x faster than HDD; flash is popular in OLTP, but not so much in OLAP or SQL data processing. Rini then took a look at the opportunity for data reduction in OLAP by looking at TPC-H Query 6. For selectivity, there are lots of irrelevant data reads. Rini then talked about flash optimized FlashQueryFile (FQF) challenges: Skipping data is intuitive in the projection phase as row IDs of interest are already known; simple random accession of data is not feasible; the same layout does not work across various column cardinalities.

Rini then introduced selection-optimized columnar data layout and projection-optimized columnar data layout in FQF. In the evaluation, Rini talked about the experimental setup. In terms of results, FQF achieved 11x–100x speedup and achieved a 38% to 99.08% reduction in data read compared with ORCFile on flash.

Margo Seltzer asked what happens when OLTP, instead of OLAP, is made flash aware. Rini replied that scalability is one issue and another issue is that the majority data of OLTP is read-only and no update is required in this case.

Hotpourri

Summarized by Cheng Li (chenglii@cs.rutgers.edu)

Assert(!Defined(Sequential I/O))

Cheng Li, Rutgers University; Philip Shilane, Fred Douglis, Darren Sawyer, and Hyong Shim, EMC Corporation

Cheng Li presented his research on revisiting the definition of sequential I/O. He first motivated the work by addressing the importance of the concept of sequential I/O, because many optimizations were made based on the concept of sequentiality to improve performance of disk or tapes. Many applications leverage sequential I/O such as caching and prefetching. In addition, many non-rotational devices such as SSDs favor sequential I/O because writes with large I/O size can reduce the number of SSD erasures, which improves SSD lifespan. Finally, a clear definition of sequential I/O helps classify workload characteristics, which will benefit the system researcher, trace analysis, and synthetic I/O generation.

Cheng showed a few definitions of sequential I/O and did a live survey with the audience, asking them which definition best matched their intuition. More people from the audience preferred the second definition of sequential I/O, but there was no consensus. Cheng suggested that sequentiality is heavily used in literature but rarely defined, and defined in an inconsistent way. Cheng used a big-data driven approach to compare different sequentiality metrics.

Cheng reviewed several definitions of sequentiality and properties of sequential I/O that might impact the sequentiality definition. First, he showed the canonical definition of sequentiality, the consecutive access ratio, defined as the fraction of consecutive accesses. Then he presented another way of measuring sequentiality, the consecutive bytes accessed. The consecutive bytes accessed incorporates the I/O size so it captures more properties compared to the simple consecutive access ratio. Then Cheng presented a strided range property that allows gaps, small backward seeks, and re-access continuations to be considered as sequential accesses. This property improves the strict definition of sequentiality. Cheng presented the multi-stream property that leverages application information to separate out mixed I/O streams and an inter-arrival property that defines consecutive I/O requests with long intervals as non-sequential.

Cheng presented the methodology of this study. He looked at the different combinations of the sequentiality properties in the definition. Then he tried to use different metrics to measure sequentiality of storage traces. He compared a ranked list produced by different metrics. If all metrics provide the same view towards sequentiality, then it doesn't matter which definition to use; otherwise, it's necessary to pick metrics that best align with the use case and study the correlation of different metrics.

Cheng presented several interesting results. The primary findings are: (1) Different sequentiality metrics provide different views towards sequentiality; (2) the metrics that incorporate I/O size show a more consistent view when quantifying access patterns; (3) many sequentiality metrics are negatively correlated, which means the results can change completely depending on which metrics to use; (4) although there might not be a global metric for sequentiality, system researchers should pick one that best aligns with the use case and state which definition to use.

Peter Desnoyers (Northeastern University) asked about what if the same application uses different metrics. Cheng answered that he looked at caching as an example, and different metrics indeed produce diverse different sequentiality values, which makes it hard to make a conclusion based on different metrics.

Towards Paravirtualized Network File Systems

Raja Appuswamy, Sergey Legtchenko, and Antony Rowstron, Microsoft Research, Cambridge

Sergey Legtchenko motivated the work by comparing VHD and NFS with emerging hardware. Then he asked, what are the tradeoffs in choosing one versus the other? Are current mechanisms sufficient with emerging hardware?

Sergey quantified the VHD overhead in the datacenter today and contrasted this with the VHD overhead in emerging datacenters. VHD causes high overhead but is fully compatible with other features. NFS incurs low overhead but is incompatible with other features. Clearly, there is a need for a new data access mechanism that can avoid nesting like NFS, and also enable hypervisor interception like VHD. So this work proposed a paravirtualization scheme.

The paravirtualized NFS client performs first-level DRAM caching and passes through cache misses to the hypervisor that acts as a proxy, while the hypervisor NFS client achieves second-level caching with flash or memory. Existing protocols, like SMB, can be used for forwarding requests to the NAS server. It is non-invasive, backward-compatible data access. There is no revisiting the guest-host division of labor.

There are several paravirtualization tradeoffs. The advantages of paravirtualized NFS client shows performance similar to NFS, feature compatibility similar to VHDs. In addition, it supports end-to-end semantic awareness. So clients can use NFS protocols for accessing and sharing data. And unlike VHD, files stored within an NFS server can be shared.

There are still challenges: e.g., implementing the low-overhead guest-host file I/O bypass, implementing file-level protocols. Still, there is a lack of full-system virtualization.

In conclusion, storage hardware is changing quickly; there is more low-latency RDMA-based access to storage class memory. There is a need for flexible, overhead-free data-access mechanisms. NFS is overhead free but incompatible with other features. VHD is compatible, but suffers from overhead due to translations. The proposed paravirtualizing NFS client is as fast as NFS, compatible as VHD. Paravirtualized NFS is non-invasive and builds on well-established protocols and interfaces.

The first question was for a clarification to avoid the confusion between the term NFS that the talk used referring to the general concept of network file systems and the NFS protocol. Sergey answered that they were using NFS to refer to a network file system, not the NFS protocol. The second question was on the experimental setup: Which version of the SMB protocol were they using and were all stacks Microsoft-based? Sergey answered that they are using SMB 3.0, which enables direct access over RDMA (SMBd), the host runs Hyper-V and the guest runs Windows Server 2012.

Evaluation of Codes with Inherent Double Replication for Hadoop

M. Nikhil Krishnan, N. Prakash, V. Lalitha, Birenjith Sasidharan, P. Vijay Kumar, Indian Institute of Science, Bangalore; Srinivasan Narayanamurthy, Ranjit Kumar, and Siddhartha Nandi, NetApp Inc.

Prakash first compared the triple replication of data in Hadoop with double replication. The Hadoop replication, while achieving high data protection, increases storage overhead significantly. Another useful scheme is the RAID6 + mirroring, which uses two parity blocks to ensure adequate resiliency.

The challenge to address was to apply inherent double replication coding schemes to improve locality for Hadoop. Prakash introduced the Heptagon-local code (HLC), which is an alternative code with inherent double replication. The HLC has reduced overhead for the desired resiliency but there is an issue relating to data locality. Clearly, it's important to leverage data locality to ensure computation is completed locally. The way they address the locality is to modify the HDFS to permit coding across files.

Prakash used several slides to explain how to build the Heptagon code, providing some insights on the Heptagon codes as a rearrangement of RAID+m. The resiliency of Heptagon code can tolerate two out of five node failures, recovered by parity. How to extend the code to a Heptagon code and how to recover from two/three node erasures and overhead/resiliency results were discussed next. Finally, Prakash discussed data locality for the Heptagon code and showed MR performance in Hadoop.

Someone asked about making a comparison with a class of error-correcting codes known as Fountain codes.

SSDelightful

Summarized by Prakash Narayanamoorthy (prakashnarayanamoorthy@gmail.com)

The Multi-streamed Solid-State Drive

Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho, Samsung Electronics Co.

Jeong-Uk Kang presented a multi-stream-based approach for improving the efficiency of garbage collection (GC) in solid state drives (SSDs). She started off by saying that SSDs share a common interface with HDDs, which facilitated faster adoption of SSDs. However, since rotating media and NAND-based SSDs are very different, such a common interface is enabled by the use of a Flash Translation Layer (FTL) in the SSDs. The FTL has two purposes; one is logical mapping of blocks and the other is to do bad-block management performed via GC, which serves to reclaim space and to erase blocks. However, in the current implementations, GC is an expensive operation and it highly affects the SSD life.

The authors presented a new approach for improving the efficiency of GC. Their idea is to create streams while writing data into SSD. The various streams are chosen based on the life expectancy of the data that is being written. Kang argued that the best performance is obtained when the lifetime of data being written is determined by the host system itself and passed on to the SSD, which then determines the stream ID. In this approach, GC can be done in a targeted manner, on the blocks corresponding to the individual streams. The idea was tested in Cassandra, using a new interface that implements up to four different streams and by using the YCSB benchmark. Performance with the "TRIM on" feature was also evaluated. The test-case with four streams showed the best performance.

Someone asked whether the approach was specific to flash, to which Kang remarked this to be general to all SSDs. As to whether modifications to Cassandra were necessary, Kang replied in the negative. Someone asked whether the SSD block layer had to be modified. Once again, Kang noted this as being not necessary.

Accelerating External Sorting via On-the-Fly Data Merge in Active SSDs

Young-Sik Lee, Korea Advanced Institute of Science and Technology (KAIST); Luis Cavazos Quero, Youngjae Lee, and Jin-Soo Kim, Sungkyunkwan University; Seungryoul Maeng, Korea Advanced Institute of Science and Technology (KAIST)

Young-Sik Lee presented a new architecture for improved in-storage processing in SSDs, which seeks to improve I/O performance and hence the life of the SSD. The idea was to use an active SSD architecture, which will perform external sorting algorithms more efficiently. Lee started off by stressing the importance of I/O in data-intensive computing and the need to migrate to SSDs to improve the I/O performance. Although in traditional SSDs, the storage and the host processor remain separate, in active SSDs, there is room for in-storage processing, which can further improve the I/O performance of the SSD.

Lee said that although there are existing architectures for active SSDs, they only perform aggregate functions (like min, max, count). However, given the increased processing power of active SSDs, more complex functions can be performed in-storage.

The new architecture considered by the authors would allow computation of non-aggregate functions. Specifically, the focus was on an operation referred to as the active-sort, which improves the efficiency of external sorting algorithms. Lee remarked that such algorithms played a major role in the Hadoop MapReduce framework. In the traditional way of sorting, the SSD stores partially sorted chunks, which are read out by the host to do the final merge. This merged output is then written back to the SSD to be used by the next stage of processing. In active sorting, the SSD simply keeps the partially sorted chunks, and when the next stage of processing demands the merged data, the merging happens inside the SSD itself; this result is fed to the next stage. Thus there are significant savings in write bandwidth, since the host need not write back the merged data to the SSD. There is, however, a small increase in read bandwidth to perform the in-storage merging. An implementation was carried out on an open SSD platform, consisting of four channels, each of 32 GB. The SORT benchmark was run on the new architecture, and measured quantities include read/write bandwidth and elapsed-time for the sort operation. A comparison was performed against the NSORT and QSORT algorithms, and gains were demonstrated, especially when the host memory was small compared to the size of the data being sorted. Lee concluded by saying that their next plan was to integrate this architecture with that of Hadoop MapReduce.

In the question and answer session, someone felt that even though there are savings in I/O for the SSDs, there might not be an improvement in its lifetime, since there are other factors affecting longevity. Another questioner wondered whether the proposed architecture could be applied in situations other than the MapReduce framework. Lee noted that this was also one of the points that they were actively thinking about.

Power, Energy, and Thermal Considerations in SSD-Based I/O Acceleration

Jie Zhang, Mustafa Shihab and Myoungsoo Jung, The University of Texas at Dallas

Jie Zhang generated a considerable amount of conversation among the audience around the topic of whether multi-resource SSDs that promise high I/O can deliver it at a reduced power and energy consumption, as is commonly believed, or whether they needed to consume energy to deliver the improved I/O performance. Zhang started off by noting that a single SSD chip has a very limited I/O rate, and it is common to use many chips to match the PCI-Express bandwidth. Modern SSDs also come with many channels and many controllers and cores to handle multiple tasks in parallel. The number of components integrated into these many-resource SSDs have increased by more than 62 times with respect to what was seen during the early 2000s.

While all these new components were added for improved I/O and latency performance, Zhang highlighted the lack of studies on the power, energy, and thermal considerations for these new many-resource SSDs.

Zhang presented many measurements to show that, contrary to popular beliefs, power, energy, and thermal properties of the new SSDs are much worse than traditional SSDs. Measurements revealed that while single-purpose SSDs measure around 95–120 degrees Fahrenheit (operating temperature), multi-purpose SSDs could go up to 180 degrees Fahrenheit during their operation. It was also demonstrated that the improved latency of the multi-purpose SSDs comes with an overhead of around seven times increased dynamic power consumption. Zhang also pointed out that due to such enormous power consumption, the internal mechanism of the SSD automatically reduces the performance in response to the heat generated. Zhang concluded that the overheating problem and power throttling issues are holding back state-of-the-art SSDs.

Someone asked whether making the SSDs byte-addressable and providing them with direct access to the memory bus would eliminate some of these power consumption issues. Zhang said that the current results may be affected by the suggested modifications. Another questioner wondered about profiling heat generation patterns of the various components and suggested studying which of the many components present in the multi-purpose SSDs contributed to the increased power consumption. Zhang noted that more measurements are needed in that direction and reserved that for future work.