Sixth Workshop on Hot Topics in System Dependability (HotDep '10)

October 3, 2010
Vancouver, BC, Canada

Distributed Algorithms

Summarized by Hussam Abu-Libdeh (hussam@cs.cornell.edu)

Storyboard: Optimistic Deterministic Multithreading

Rüdiger Kapitza, Matthias Schunter, and Christian Cachin, IBM Research—Zurich; Klaus Stengel and Tobias Distler, Friedrich-Alexander University Erlangen-Nuremberg

Rüdiger Kapitza opened his talk by noting how nowadays conventional infrastructure is replaced with network-based services where redundancy via state machine replication is used to balance load and achieve high availability. In a typical deterministic state machine replication setting, clients talk to the replicated service via agreement nodes that produce an ordering among client requests, which are then forwarded to execution nodes. As a consequence, we expect that every non-faulty replica will produce the same non-faulty output for the same sequence of client requests. Even though this sounds simple, multi-threaded execution at the replicas complicates things by introducing nondeterminism due to scheduling.

To solve this issue, Rüdiger introduced the Storyboard design for lock prediction and controlled execution, where an oracle is used to predict replica concurrency issues by representing execution paths as ordered lists of lock accesses, which are then executed in a controlled multi-threaded fashion.
In the Storyboard design, clients talk to agreement nodes, which then talk to predictor nodes that predict and forecast locks usage, and finally a controlled execution is carried out by the replica nodes which operate according to the forecast Storyboard. In a controlled execution, threads are allowed to execute at their own speed, but they are only allowed to enter into the predicted list of critical sections and are not allowed

to overtake other threads into a critical section, and thus the execution will follow the forecast "story."

This was the main idea of Storyboard, and Kapitza proceeded to talk about implementation issues such as handling mispredictions and complex locking structures such as condition variables to synchronize multiple threads, and nested locks. Development is currently underway for a Storyboard prototype, and preliminary results showed an analysis of lock usage in CBASE-FS.

After the talk an audience member asked whether execution rolls back if a thread needs to take an unpredicted lock. Kapitza answered that there is no need for rollbacks, since the current system pauses the execution at the point where an unpredicted lock is requested, and that will enforce the new story across the different replicas. In response to another question, the presenter acknowledged that the current Storyboard design does not address situations with data races, although that is a point for future work.

Scalable Agreement: Toward Ordering as a Service

Manos Kapritsos, UT Austin; Flavio P. Junqueira, Yahoo! Research

Common practice in reliable services is to deploy replicated execution nodes that are preceded by ordering nodes that order client requests so that replicas execute in the same order. However, ordering is left for service developers to deploy, which requires us to provision for nodes that do not do computation and that can additionally become a bottleneck if the core service becomes popular. With this setting, Manos presented his vision for request ordering as a utility service.

A problem is that ordering uses agreement protocols which do not scale, and in fact, generally, adding more machines to agreement protocols increases complexity and not throughput. This is because in most agreement protocols, clients contact a primary node that proposes an order for the requests and broadcasts it to replicas that do an all-to-all communication to agree on the order and finally execute.

Scalable ordering protocols are needed to enable Ordering-as-a-Service, and here Manos proposes leveraging multiple small ordering clusters to compute partial orders and using virtual slot space to get the full order. In virtual slot space each ordering cluster is assigned a color, and the full order is composed by interleaving a slot from each color. For example, given three ordering clusters—blue, red, green—a full ordering schedule can be obtained by executing the first request from the blue cluster first, followed by the first request from the red cluster, followed by the first from the green. Next comes the second request from the blue cluster followed by the second from the red, and so on.

After explaining the general idea, Manos described implementation details such as mapping clusters to physical nodes to balance load, keep faults independent, exploit multicore, and reduce bottlenecks. A preliminary evaluation was made on Emulab with a micro-benchmark of reading/writing key-value pairs from a hashtable. The evaluation demonstrated the scalability of the ordering service by adding more nodes to the configuration and achieving higher ordering throughput. The scalability was measured in terms of adding more clusters and more machines to a fixed set of clusters.

One attendee wondered whether it would make more sense to consider correlation at the rack level rather than on a per-core/process level, and Manos acknowledged that failure correlation depends on the actual deployment environment and the availability required of the service. Hakim Weatherspoon from Cornell University asked about the performance of a service in case of a failure, and Manos responded that a machine's failure will require reconfiguration and reallocation of replicas to other machines. Finally, Atul Singh from Princeton University questioned the usefulness of having a single common ordering service for applications that do not share data, as this extra service might not be all that crucial to the applications. Manos answered that their motivation is to allow scalability such that ordering is never an issue. An added benefit of having an ordering service is that it is one less thing for developers to worry about; just use "the ordering service" and you're done.

Active Quorum Systems

Alysson Bessani, Paulo Sousa, and Miguel Correia, University of Lisbon, Faculty of Sciences

Alysson argues that state machine replication is conceptually simple and it usually provides linearizability, which is a stronger consistency model that is not required in many applications. This makes it difficult to implement tasks like housekeeping/cron jobs, asynchronous messaging, or multi-threaded services. The current mantra is that strong consistency should be avoided at all costs and that led us to embrace eventual consistency. However, eventual consistency is not always adequate and some applications just require strong consistency.

The research question posed is, would it be possible to build dependable and consistent services that rely on strong synchrony only when it is absolutely necessary? To answer this question, Alysson looked at high-level abstractions like coordination services, and low-level abstractions like read/write quorum systems, leader election, and barriers.

Along those lines, Alysson proposes Active Quorum Systems (AQS), which he describes as a Byzantine quorum system with synchronization power. AQS breaks the system state into small objects, where instead of having the entire service as a replicated state machine, the service is viewed as a set of replicated objects. AQS supports three types of low-level operations: read, write, and read-modify-write, which updates the state of an object using its old value. Read and write operations are implemented as in typical quorum-based asynchronous protocols. The read-modify-write operation is implemented as an extension to PBFT (Practical Byzantine Fault Tolerance) where the primary acts locally on a received request and then broadcasts the triple (start state, command, result) to all the replicas. If the replicas approve, then the change is committed; otherwise the most recent copy of the state is sent back to the primary and the operation is repeated until consensus is achieved. A final design principle of AQS is that the service specification is exploited in order to find opportunity for optimization (so not based on the environment, because it can change). An example of that is determining the level of consistency by the service needs, and the same goes for writer access control. Alysson argued that the benefits of AQS are that it makes minimal assumptions, achieves communication optimality, and provides stability for non-favorable executions.

In response to a question from the audience, Alysson noted that unfortunately AQS adds complexity to building systems for non-experienced users. Hakim Weatherspoon asked what would happen if the service assumptions were violated. For example, what would happen if the system were to evolve to allow multiple writers? Alysson responded that the idea of having multiple writers is not associated with contention but with access control. They in fact encountered a case of evolving the system when working on LDAP but it has not been completely worked out.

OS Reliability

Summarized by Mark Spear (mspear@cs.ubc.ca)

We Crashed, Now What?

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

Cristiano Giuffrida presented an operating system model that addresses many problems involved in crash recovery. He immediately brought laughs to the audience with a Blue Screen of Death slide; in this particular BSOD, a device driver bug brought down the entire operating system. Much of the related work on crash recovery focuses on isolated subsystems (e.g., device drivers, file systems). In these works, a portion of the system is trusted to monitor the untrusted portion (e.g., the driver). But when extending crash recovery to the entire system, that model would require monitoring the monitor, ad infinitum, "like a dog chasing its tail."

Instead, Cristiano's group elected to combine OS design and lightweight instrumentation to scale crash recovery to the entire OS. They break down the operating system into independent user-space processes, resulting in a multiserver microkernel-based OS architecture. Each process follows an event-driven model and is solely dedicated to carrying out a specific task in a loop, termed the task loop. By design, the top of the task loop is a local stable state. The task loop can generate idempotent messages throughout, and any non-idempotent messages generated while handling a request are pushed to the end of the loop. Lightweight recovery code is added through instrumentation by LLVM and is used to revert to the last stable state in the event of failure. A shadow state region is used and memory allocations are tracked, as are object state changes, etc. These changes are all committed at the top of the event loop. When the system manager detects a crash (e.g., in the Process Manager component), a replica of the component has the last stable state transferred to it and resumes operation as if nothing bad happened. The system manager then cleans up the dead component. The authors have prototyped the ideas described in the paper on top of the MINIX 3 microkernel.

One audience member voiced concerns about several issues, including multi-threaded servers, communication through shared memory, and blocking on I/O. Cristiano responded by saying that they are not aiming for backward compatibility but, rather, designing a new system using the event-driven model. This model would use asynchronous IPC instead. The performance evaluation was questioned, as this system was compared to unmodified MINIX 3 rather than Linux. Cristiano called attention to the fact that the scalability graph was normalized data, only showing relative performance/overhead. How long did this work take? It was hard for Cristiano to separate the time involved for different parts, but he said that about one year was spent on implementation, with previous work already having been done on the design.

Improved Device Driver Reliability Through Verification Reuse

Leonid Ryzhyk, NICTA and University of New South Wales; John Keys, Intel Corporation; Balachandra Mirla, NICTA and University of New South Wales; Arun Raghunath and Mona Vij, Intel Corporation; Gernot Heiser, NICTA and University of New South Wales

Leonid Ryzhyk observed that while hardware device verification and device driver development have a remarkable degree of similarity, the two processes are currently completely disjoint. Thus we are robbed of an opportunity for more reliable driver development. Leonid's presentation began in the same fashion as the previous one, with a Blue Screen of Death, and the audience continued to find this gag funny.

Current techniques for dealing with driver reliability include runtime isolation, static analysis and model checking, safe languages, etc. At the end of the day, drivers are still much less reliable than we'd like, and Leonid proposes a complementary approach for improving driver reliability. He presented the observation that the most common class of driver bug is device protocol violation (e.g., using an invalid sequence of commands, wrong use of DMA descriptor, interpreting data incorrectly). Hardware designers communicate to the driver developers through a datasheet, which often contains inaccurate information. However, the hardware verification engineers are privy to more details about the device, and lots of effort is put into the verification testbench. The testbench has several layers, including scenario, agent, and the device under test, which have analogs in the OS I/O stack. The scenario layer should be extended to be OS-based and simulate how the OS uses a driver. The agent layer has a similar role to that of a driver (translating a high-level request to low-level operations, and changing state of the device). Leonid suggests that an actual driver could replace the agent layer, so that hardware and software are being co-verified. In order to facilitate cross-platform use of the single driver under test, Leonid proposes unified driver interfaces (per device class), instead of naively emulating existing OS interfaces in the testbench (which would lead to OS-specific testing). The resulting driver could then be used without modification in real operating systems.

This could result in several benefits, including a reduced dev cycle and (naturally) fewer bugs in the end product. They found a number of defects in USB and UART drivers, an Ethernet hardware race condition, and several other bugs that weren't found using the conventional development method.

A number of questions involving different testing configurations were asked. The first inquiry was about different versions of a hardware device. That could require updated testbenches in both the conventional and the proposed model. The same questioner also asked about different versions of a driver. The response was that drivers would require testing anyway, so now it would be done through the testbench. When an operating system changes, is it the job of the hardware vendor to retest the driver? It would require work if the testbench was emulating the OS, but instead a generic interface is assumed, so that isn't an issue.

Towards Automatically Checking Thousands of Failures with Micro-specifications

Haryadi S. Gunawi, University of California, Berkeley; Thanh Do, University of Wisconsin, Madison; Pallavi Joshi and Joseph M. Hellerstein, University of California, Berkeley; Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison; Koushik Sen, University of California, Berkeley

Thanh Do presented a mechanism for exploring complex failures and error recovery scenarios. In the era of cloud computing and using thousands of machines, failures are not so rare anymore. The reliability has to come from the software, but even the big players get it wrong sometimes (e.g., Sidekick data loss, Facebook photo loss). Thanh argues that current testing of failure recovery is insufficient. In response to this problem, his group has developed a pair of tools that work in concert to explore failures. In the time since their paper was submitted to HotDep, these tool, FTS (Failure Testing Service) and DTS (Declarative Testing Specification), have been renamed FATE and DESTINI, respectively.

FATE is a failure injection framework. It targets I/O points and can exercise many combinations of failures. A "Failure ID" is their representation of a failure: It contains a failure point (the system/library call that performs disk or network I/O), a failure type (crash, exception, etc.), a stack trace, and some domain-specific information (e.g., source, destination, message). The hash of a failure ID is used to log the failure history when exploring the failure space. Aspect-oriented programming (AspectJ) is used to instrument Java programs with no changes required to the system under test. Multiple failures are injected, including failures during recovery. DESTINI is responsible for testing whether actual behavior is consistent with what was expected. Violations are detected through evaluation of Datalog style rules.

The authors applied their system to three cloud systems, HDFS, ZooKeeper, and Cassandra, writing 74 recovery specifications at an average length of 3 lines per specification. Their system found 16 bugs and reproduced 74. The bugs caused reduced availability and performance, data loss during multiple failures, and errors in the recovery protocol.

The first questioner pointed out that one man's bug is another man's feature: What do you do when you don't have a precise specification? Thanh noted that even without a precise specification, you should at least have a high-level expectation of how the system works. If necessary, the specification could be refined later, and there is no need to start at a low level. Another questioner asked about one of the hardest situations to test: arbitrary corruption. The framework supports various kinds of corruption (e.g., network packets), but the full answer about arbitrary state corruption was taken offline. Finally, an audience member noted that when considering multiple possible failures, the state space explodes. Thanh mentioned that they were looking at heuristics for pruning the failure space, prioritizing some failures, and focusing on "interesting points," and that some of that work was submitted to NSDI.

Management and Debugging

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

Focus Replay Debugging Effort on the Control Plane

Gautam Altekar and Ion Stoica, UC Berkeley

Gautam Altekar began his presentation with the observation that debugging datacenter software is particularly difficult, for three reasons: it is large-scale, data-intensive, and non-deterministic. Altekar argued that static techniques do not scale to the state space of these large systems, and so we need a way to do deterministic recording and replay of production systems in order to reproduce nondeterministic bugs. But production systems will not tolerate a great deal of overhead in either performance or logging data rate.

Altekar proposed that datacenter applications will typically have two somewhat distinct components: a control plane for managing data flow and maintaining replica consistency, and a data plane of relatively simple data processing engines. He then hypothesized that the control plane, being complex, would have a much higher relative bug rate than the data plane. At the same time, it would have a much lower data rate. For reproducing bugs in the control plane, which he argued were the most important and difficult, Altekar claimed that it would suffice to maintain deterministic recordings of the control plane.

To test this hypothesis, he chose three applications (Hypertable, KFS/CloudStore, and OpenSSH) and used taint tracking to classify code that accessed user data as data plane code (this classification needed manual refinement because of the high rate of false positives produced by taint tracking at the CPU level, and because classification based on observed execution had poor coverage). Bearing in mind that the results were not very scientific, his initial analysis appeared to justify his hypothesis: 99% of the bugs reported in these applications were in "control plane" code, but this code accessed only 1% of the data processed during execution. Altekar believed that this result warranted further investigation.

There were a lot of questions about how cleanly control and data plane code could be separated in practice. Derek Murray (Cambridge) wondered how this would work for systems like Google Percolator, in which the results of the data plane could affect the control plane. Dutch Meyer (UBC) asked whether data traffic could be distinguished by directly examining the data. Altekar responded that the distinguishing feature of data plane data was volume. Steve Hand (Cambridge) noted that the chosen applications were application frameworks and wondered how well the observed bug rates would correspond with those for actual applications built on the frameworks. Altekar intended to look into that.

A Rising Tide Lifts All Boats: How Memory Error Prediction and Prevention Can Help with Virtualized System Longevity

Yuyang Du and Hongliang Yu, Tsinghua University; Yunhong Jiang and Yaozu Dong, Intel Research and Development, Asia-Pacific; Weimin Zheng, Tsinghua University

Yuyang Du began his talk with the surprising claim that RAM errors cause the plurality of system failures (including both software and hardware causes). While these could be prevented with hardware fault tolerance or even simple ECC, Du claimed that cost prevented widespread deployment of these techniques. He further noted that in the increasingly important cloud hosting economy, consumers don't have direct control over the hardware on which their applications run. He also claimed that virtualization compounded the problem of unreliable memory, because the physical hardware was multiplexed across multiple servers.

Du noted that memory chips produced both correctable (soft) and uncorrectable (hard) errors. He proposed that for virtualized servers, a cost-effective approximation of ECC could be achieved by using observed soft errors to predict future hard errors and migrate virtual memory off the failing physical RAM before they happened. He had not yet evaluated his failure predictor or any gains in reliability versus the cost of disabling physical RAM. He noted that this sort of project was very slow to test, since it depended on waiting for memory to fail.

The questioning focused on two topics: the accuracy of Du's hard memory error predictor, and on the idea of creating graduated memory protection. Steve Hand (Cambridge) asked about using ECC memory for the hypervisor but only best-effort, predictive protection for virtual machines. Karthik Pattabiraman (UBC) took the idea further, proposing that individual applications might be able to make use of memory of varying reliability. Du agreed that these were interesting avenues of research.

A Design for Comprehensive Kernel Instrumentation

Peter Feiner, Angela Demke Brown, and Ashvin Goel, University of Toronto

Peter Feiner and his co-authors would like to protect systems from faulty device drivers, using techniques like Microsoft Research's Byte Granularity Isolation. The trouble is that such techniques require source code, but often the source for device drivers is unavailable. Applying this technique to binary code requires a dynamic binary instrumentation (DBI) framework (like Valgrind, DynamoRIO or Pin), but Feiner claims that no such framework exists that can operate on kernel code.

The authors see two approaches to constructing a kernel-level DBI framework: either port an existing tool like Pin to an existing hypervisor, or build a minimal hypervisor with just enough code to support DBI. Claiming that a port would be difficult due to the amount of code that would need changing, they instead investigated the features required of a custom DBI hypervisor. Feiner spent the rest of his talk enumerating many of the tricky issues involved in performing invisible code translation at the kernel level.

I was surprised both by the claim that a port approach was more difficult, and that it hadn't been done before. I asked Feiner how his model compared to PinOS, which combined Pin with Xen and was described in a VEE paper in 2007. Feiner said that the disadvantage of PinOS was that the TCB for the translation engine was much larger. Others asked about overhead, and issues dealing with self-modifying code. Feiner noted that his talk was about a proposed architecture rather than an implemented system, and so discussions of mechanics and performance were not yet relevant.

Storage and File Services

Summarized by Mark Spear (mspear@cs.ubc.ca)

Behavior-Based Problem Localization for Parallel File Systems

Michael P. Kasick, Rajeev Gandhi, and Priya Narasimhan, Carnegie Mellon University

Mike Kasick described a method of diagnosing problems in parallel file systems by analyzing system behavior via CPU instruction pointer sampling and function call tracing. This work was motivated by real problems experienced by the developers of PVFS (Parallel Virtual File System): limping-but-alive servers that reported no errors, faulty and overloaded switches, buggy RAID controllers, and a variety of other problems that may pass their respective diagnostic tests. Previous work has shown instances where performance manifestations of problems were masked by normal deviations. However, behavioral manifestations may be more prominent than performance manifestations.

Fault-free peers have similar behavior: e.g., large I/O requests are striped across all servers, and small I/O requests, in aggregate, also equally load all servers. A system exhibiting a fault (e.g., "Write-Network-Hog Fault") will have a behavioral manifestation (e.g., a gross discrepancy in calls to the kernel tcp_v4_rcv function). Each server has a feature vector of several metrics from a sliding window of time. The features include samples, function call counts, and time. To detect anomalous behavior, Manhattan distances between feature vectors are computed pair-wise between servers. The median distance for a given server is tested against a threshold: The (per-server) threshold for "anomalous" is set via a fault-free high-stress training phase, to find the maximum deviation expected under normal conditions. This requires training on each cluster/filesystem combination, but not based on workload.

Examining the most anomalous metrics facilitates root-cause analysis. Given an indicted node, and its feature vectors over time compared to others, the system can present a list of possible anomalies for manual inspection. Different metrics are good at different kinds of faults. Disk faults are best detected by the time metric, since blocking I/O calls are used. Count metrics are good for detecting packet loss: dropped packets cause more non-blocking reads, resulting in a higher function call count. Sampling is useful for detecting the "network hog" fault, as TCP retransmits increase the CPU load.

Steve Hand (Cambridge) asked a question about how the threshold was selected. It appeared to be a constant number; why not set the threshold as a function of what is observed at runtime (like a number of standard deviations)? Mike noted that it is not really an absolute number, although the example slide may have made it seem that way. Instead, it is a maximum degree of tolerable deviation.

What Consistency Does Your Key-Value Store Actually Provide?

Eric Anderson, Xiaozhou Li, Mehul A. Shah, Joseph Tucek, and Jay J. Wylie, Hewlett-Packard Laboratories

Xiaozhou (Steve) Li presented an analysis of key-value stores. Service-level agreements for consistencies of key-value stores are likely on the horizon. Most stores only promise eventual consistency, but for some workloads (depending on the number of updates, how much contention there is), it may perform better.

Key-value stores are commercial black boxes. What can you do to analyze the consistency to see if you need a higher level of service? Client machines can record sequences of get/put requests and record the time that requests are sent and replies are received. Based on the observed sequence, one can attempt to analyze atomicity, regularity, and safety (properties from Lamport's work on register-based consistency). The presentation focused on atomicity, and a graph theoretical approach was offered. The vertices represent operations and edges represent precedence. The sequence is a "good" sequence if and only if it is a directed acyclic graph. A cycle would imply that a vertex should happen before itself (because edges are precedence). There are three types of edges: time (if an operation precedes another entirely), data (write 0->read 1; the value of the write should appear in the value of the read), and "hybrid" edges (which enforce the invariant that "all writes time-preceding a read should happen before the read's dictating write"). Cycles are counted via DFS, and the number of cycles detected is representative of how severely inconsistent a trace is.

Measurements of the same sequence of operations from the service provider's side are necessarily shorter, because they wouldn't include network latency. Therefore, more time edges would be added to the graph (from the service provider's perspective). Thus, a user detecting a violation (i.e., a cycle in the graph) would imply that the service provider would also know there was a violation. Some evaluation was done, and it was noted that their key-value store, Pahoehoe (which is eventually consistent), with sufficiently low contention, is about atomic.

Steve agreed with an interesting possibility presented by an audience member: With these commercial systems exposing the same interface, if violations are noticed, one could take action and switch to another provider. Another audience member asked, if a service provider violation is detected, how can you prove it? Steve noted the service provider, if running this analysis, would also detect the violation, because they would have more time edges (because of the apparent shortening of operations). But to actually "prove" the violation, you would need to incorporate non-repudiation techniques to convince a third party, using digital signatures and related techniques. The final question was whether the algorithm could be run in parallel if keys were in disjoint cliques, and the answer was yes: it is straightforward, as if they are different keyspaces.

ing with the guard band is much worse than the nominally achievable results.

Puneet advocates that exposing aspects of this variability to software can allow improved functionality. Such exposure could happen via active measurement or prior testing, but it can have a significant impact. For instance, in sensing applications the sleep power is dominant and can affect the amount of data that can be acquired on a given power budget. The authors found that for 10 off-the-shelf Cortex M3 processors, the active power varied by 10%, but the sleep power varied by 80%. Furthermore, the sleep and active power vary with temperature. Using a power model calibrated to temperature and sensors for power, Puneet demonstrated that they can achieve more effective sensing by energy-aware duty cycling. Thus, a node with lower sleep power can sample 1.8x more data than it would have if all nodes were timed to the worst sleep power. Moving forward, one challenge is to discover the right interface for exposing the variability and sense data to software.

An audience member observed that this data is already integrated in modern CPU power management units for managing the frequency within temperature and power bounds. Puneet responded that it is important to actually expose this information to the software layer, which most of these techniques fail to do. Another audience member asked about the overhead of these techniques. Puneet observed that the information is already being collected for quality control but it is not exposed to software. Finally, an audience member asked about the difficulty managing other system components. Puneet said that the complexity would depend on the abstraction.

Invited Talk

Summarized by John McCullough (jmccullo@cs.ucsd.edu)

Datacenter Power Efficiency: Separating Fact from Fiction

Kushagra Vaid, Microsoft Corporation

Kushagra Vaid posed the question of how to maximize power efficiency at a large scale. A data center consists of a power substation, chillers, batteries, an ops room, generators, fuel, and computing equipment. The efficiency of a facility is often measured by PUE, which is the facility power divided by the IT equipment power. Common PUE values range around 1.5–2.0, but good deployments reach close to ideal at 1.05.

Kushagra observed that the cost of power is fairly low relative to the overall costs of the facility. Amortizing using a three-year replacement model, power consumption consists of 16% of the cost. Thus, reducing dynamic power and energy-proportional computing has limited benefits. The capital expense of servers accounts for the largest portion of the total cost of ownership.

To minimize costs in provisioning, Microsoft is moving towards modular data centers. A video showing the modules is available at http://www.microsoft.com/showcase/en/us/details/84f44749-1343-4467-8012-9c70ef77981c. The modules function using adiabatic cooling with outside air and only using top-of-rack fans to adjust for cooling/heating as appropriate. To reduce the cost of power, there are techniques for eliminating conversion steps. Surprisingly, running AC to the servers is not significantly different from DC in total conversion efficiency. Right-sizing for density suggests a sweet spot, with the lowest-voltage processor getting a surprising performance/watt/cost benefit. Right-sizing storage can be achieved by in-depth storage trace analysis to understand workload patterns and dynamic range to pack better.

Overall, researchers need to be sure that they take a holistic view. Current research areas are in optimal provisioning for high dynamic range workloads, addressing energy proportionality via system architecture innovations, power-aware task scheduling on large clusters, and energy-conscious programming using controlled approximation.

One audience member asked why their efficiency approaches don't work in all data centers. Kushagra responded that getting all of the layers of UPSes and voltage conversion requires control of the entire data center, which few have the scale to accomplish. What are the implications of low-power CPUs in data centers? For Bing workloads, Xeon systems are 2.3x better in performance/watt/cost. Someone asked why they don't turn off servers. Kushagra replied that it can lead to response time spikes, that turn-on time can be long, and that if you can turn off servers, you brought too many online or you are doing poorly at task scheduling.

Data Center I

Summarized by John McCullough (jmccullo@cs.ucsd.edu)

Analyzing Performance Asymmetric Multicore Processors for Latency Sensitive Datacenter Applications

Vishal Gupta, Georgia Institute of Technology; Ripal Nathuji, Microsoft Research

Asymmetric multicore processors (AMPs) are being explored in multiple dimensions, where cores are combined with different performance characteristics and deployed with different functional characteristics. Vishal Gupta presented their technique for understanding the impact that AMPs can have on data centers with respect to power and throughput.

Vishal described two use cases: energy scaling and parallel speedup. Energy scaling involves execution on a combination of the small and large cores to achieve the same computation within a deadline at lower power. Parallel speedup occurs when a larger core on an AMP can execute serial sections faster. To ascertain the effects of these use cases, Vishal treats each processor as an M/M/1 queue, which models processing times as an exponential distribution where the processing time is parameterized in proportion to the chip area. Overall completion time is parametrized by the paralellizable fraction of the code. Using this model, Vishal finds that for a higher fraction of parallelizable work, power savings increase with AMP use. While there are practical considerations, AMPs offer more potential for parallel speedup than for energy speedup.

Energy Conservation in Multi-Tenant Networks through Power Virtualization

Srini Seetharaman, Deutsche Telekom R&D Lab, Los Altos

Networks are typically power oblivious and it is hard for network users to ascertain the impact. By packing flows into fewer devices and turning off unused devices, the system can turn off individual ports and even entire switches. Given a multi-tenant data center, how can tenants be influenced to reduce their system usage? Switching from a flat rate for networking to a power-based price can incentivize power savings.

Srini Seetharaman proposed the idea of virtual power. Because network power is not proportional usage, the most intuitive definition for virtual power is to split the power consumption of a component over all sharing tenants. This has the effect of penalizing a tenant for being the only occupant and encourages reuse of pre-paid/pre-powered-on elements. An implementation is in progress but there are no results yet. The specific methods of billing and pricing can influence the outcome; for instance, auctions might introduce different behavior than allocations that degrade over time. In the future, the question is how we can achieve good performance while conserving power.

In the Q&A, Srini clarified that it makes more sense to conserve in a reactive mode, turning on devices as necessary. One audience member asked whether rate-based power differences can affect power. Srini replied that the power differences are typically small. The same person also asked whether the placement of tenants in the data center could penalize them in terms of the available pricing. Srini replied that this was a concern.

Data Center II

Summarized by Etienne Le Sueur (elesueur@cse.unsw.edu)

Energy Savings in Privacy-Preserving Computation Offloading with Protection by Homomorphic Encryption

Jibang Liu and Yung-Hsiang Lu, Purdue University

Yung-Hsiang Lu presented this paper, which discusses the issues that arise when compute-intensive tasks are offloaded from mobile devices to a centralized server. The main issue their work addresses is that of privacy, when sensitive data needs to be transferred off the mobile device, using a public network, to a server which may not be trusted.

Their work mitigates this privacy issue by protecting the data using a homomorphic encryption algorithm. Homomorphic encryption is unlike traditional encryption techniques in that computations can be done on the cipher-text itself rather than being decrypted first. This way, the server operating on the data need not actually know what information the data contains.

The use-case they describe in the paper deals with image-matching—for example, when a user of a mobile phone takes a photo and wants a remote server to do some analysis to try and determine what objects the photo contains. They used an iPad portable device and a server with a 2GHz CPU for processing the images, with evaluation based on how many positive matches were made. Using their modified Gabor filter, they were able to get a correct match approximately 80% of the time when the analysis was performed on the cipher-text.

The work seems promising, and a future direction was clearly given which will address the issues with noise in the encryption system.

An audience member asked whether there were other classes of functions where it makes sense to offload computation. They haven't reached a point where there's a general rule for when to apply the technique. How strong is the encryption and can it easily be broken? The strength of the encryption depends on the key length. The discussion was continued offline.

Green Server Design: Beyond Operational Energy to Sustainability

Jichuan Chang, Justin Meza, Parthasarathy Ranganathan, Cullen Bash, and Amip Shah, Hewlett Packard Labs

Justin Meza presented this paper, which discusses a way to quantify sustainability when designing data centers. The usual motivations for the work were given: e.g., reduction of carbon footprint, secondary costs (power and cooling), and government regulation.