Conference Reports

In this issue:

For the complete 2012 USENIX Annual Technical Conference report and summaries from HotCloud '12, HotPar '12, HotStorage '12, and the panel at our first Women in Advanced Computing Summit, visit: www.usenix.org/publications/login.

2012 USENIX Annual Technical Conference (ATC '12)

Boston, MA
June 13-15, 2012

Opening Remarks

Summarized by Rik Farrow (rik@usenix.org)

Gernot Heiser (University of New South Wales) opened ATC by telling us that there had been 230 paper submissions, up by 30% from last year. Forty-one papers were accepted, after three reviewing rounds. Gernot reminded the audience that both OSDI and HotOS were coming up soon. Then Wilson Hsieh (Google) announced the best papers: "Erasure Coding in Windows Azure Storage," by Cheng Huang et al., and "netmap: A Novel Framework for Fast Packet I/O," by Luigo Rizzo.

Cloud

Summarized by Brian Cho (bcho2@illinois.edu)

Demand-Based Hierarchical QoS Using Storage Resource Pools

Ajay Gulati and Ganesha Shanmuganathan, VMware Inc.; Xuechen Zhang, Wayne State University; Peter Varman, Rice University

Imagine you are an IT administrator and your CIO asks that "all storage requirements be met, and when there is contention, don't allow the critical VMs to be affected." To get predictable storage performance, you could (1) overprovision, (2) use storage vendor products that provide QoS, or (3) provide QoS in VMs. This work looks at option 3, the goal being to provide better isolation and QoS for storage in VMs, using storage resource pools.

Ajay Gulati said that existing solutions specify QoS for each individual VM, but this level of abstraction has some drawbacks. Basically, virtual applications can involve multiple VMs running on multiple hosts—thus the need for a new abstraction, storage resource pools. Ajay reviewed an allocation model based on controls of reservation, limit, and shares.

Storage resource pools are placed in a hierarchical tree, with resource pools as intermediate nodes, and VMs at the leaves. The controls are defined on the pools as well as individual VMs. For example, the sales department and marketing department can be different resource pools with separately defined controls. These controls can be defined per-node, depending on parent, and the system can normalize these controls across the entire tree. In reality, a single tree is not used; rather, the tree is split up per datastore, but this was not detailed in the talk.

The system needs to periodically distribute spare resources, or restrict contending resources, among children in the tree, depending on the current system usage. This is done with two-level scheduling—first, split up the LUN queue limit between hosts; second, apply the queue limits by setting mClock at each VM. This is accomplished by two main steps: first, computing the controls per VM, based on demands, and second adjusting the per-host LUN queue depth. This is done every four seconds in the prototype. A detailed example of this on a small tree was presented.

A prototype was built on the ESX hypervisor, involving both a user-space module and kernel changes. Experiments were done with settings of six and eight VMs running different workloads. The results show timelines of throughput (in IOPS) per each VM, before and after changes to the controls. In summary, the system is able to provide isolation between pools, and sharing within pools.

There were no questions following the presentation.

Erasure Coding in Windows Azure Storage

Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin, Microsoft Corporation

▸ *Awarded Best Paper!*

Huseyin Simitci and Cheng Huang presented this paper together, with Huseyin starting. In Windows Azure Storage, the large scale means failures are the norm rather than the exception. In the context of storage, one question is whether to use replication or erasure coding (EC). With replication, you just make another copy, while with EC, you add parity. On failure, with replication you just read known data, while with EC you recreate the data. Both tolerate failure, but EC saves space, or can allow improved durability, with the same amount of space.

The Windows Azure Storage stream layer is an append-only distributed file system. Streams are very large files, split up into units of replication called extents. Extents are replicated before they reach their target size; once they reach the target, they are sealed (become immutable) and then EC is applied in place of replication. Previously, they used Reed-Solomon 6+3 as the conventional erasure coding: a sealed extent is split into six pieces, and these are coded into three redundant parity pieces. Huseyin concluded his part of the talk with a brief overview of practical considerations for EC.

Cheng focused on how Azure further reduces the 1.5x space requirement without sacrificing durability or performance. The standard approach to reduce the space requirement is to use Reed-Solomon 12+4 to decrease it to 1.33x. However, this makes reading expensive, and many times reconstruction happens during the critical path of client reads. It is better to achieve 1.33x overhead while only using six fragments. The key observation used to do this is the probability of failures. Conventional EC assumes all failures are equal, and the same reconstruction cost is paid on failure. However, for cloud storage, the probability of a single failure is much higher than that for multiple failures. So the approach taken is to make single failures more efficient. A 12+2+2 local reconstruction code (LRC) was developed. There are two local parities for each section of six fragments, and two global parities across all 12 fragments. In terms of durability, LRC 12+2+2 can recover from all three failures, and 86% of four failures. So the durability is between EC 12+4 and 6+3, which is "durable enough" for Azure's purposes.

LRC is tunable. You can tune storage overhead and reconstruction cost, given a hard requirement of three-replication reliability. Both Reed-Solomon and LRC are plotted as curves with the axes of reconstruction read cost vs. storage overhead. LRC gives a better curve, and the particular variant can be chosen looking at this curve. In the end, Azure chose 14+2+2, which, compared to Reed-Solomon 6+3, gives a slightly higher reconstruction cost (7 vs. 6) but has a 14% space savings, from 1.5x to 1.29x. Given the scale of the cloud, 14% is a significant amount.

Yael Melman, EMC, asked what happens when all three failures are in a single group. Cheng clarified that this does indeed work for all three failure cases, and that there are proofs of this in the paper. Richard Elling, DEY Storage Systems, asked how to manage unrecoverable reads. Cheng clarified that the failures discussed are storage node failures, not disk failures. Hari Subramanian, VMware, asked how to deal with entire disk failures. Huseyin answered that data is picked up by all remaining servers. Hari asked whether failing an entire node for a failed disk is less efficient. Huseyin clarified that entire nodes are not failed in this case, but rather that the granularity of failures considered is indeed disks. Someone asked about the construction cost when creating parity blocks—particularly the bandwidth cost involved. Cheng answered that the entire encoding phase is done in the background, not on the critical path. So you have the luxury of scheduling them as you like.

Composable Reliability for Asynchronous Systems

Sunghwan Yoo, Purdue University and HP Labs; Charles Killian, Purdue University; Terence Kelly, HP Labs; Hyoun Kyu Cho, HP Labs and University of Michigan; Steven Plite, Purdue University

Sunghwan Yoo began with an example using a KV store as motivation. He showed that many failures can happen in the chain of forwarding a request. The techniques used to mitigate these failures are retransmission, sequence numbers, persistent storage, etc. A single development team working on the whole system could make an effort to handle failures, but what if each component was handled by different teams and systems? Guaranteeing global reliability between independently developed systems is hard.

This motivates the development of Ken, a crash-restart-tolerant protocol for global reliability when composing independent components. It makes a crash-restarted node look like a slow node. Reliability is provided by using an uncoordinated rollback recovery protocol. Composability allows components to be written locally and work globally. An event-driven framework allows easy programmability—specifically, it is transparently applicable to the Mace system. These ideas (especially rollback recovery) are not in themselves new; Ken is a practical realization of decades of research.

When Ken receives a message from outside, an event loop begins—within this handler, the process can send messages and make changes to the memory heap. When the handler is finished, a commit is done, storing all changes made to a checkpoint file. An externalizer continually resends messages, to mask failures, making them look like slow nodes.

Another example was given, consisting of a seller, buyer, auction server, and banking server. If any of these systems show crash-restart failures, there are problems. Then Ken was illustrated in more detail. A ken_handler() function gets executed in a similar way to a main() function. Transaction semantics are given within the function. Calling ken_malloc()/ken_set_app_data()/ken_get_app_data() allows use of the persistent heap, while ken_send() provides "fire and forget" messages. Ken can be used in Mace without any changes. Ken provides global masking of failures, and composable reliability, while Mace provides distributed protocols, availability, replication, and handling of permanent failures.

The evaluation consists of micro-benchmarks and an implementation of Bamboo-DHT on 12 machines. The micro-benchmarks show that latency and throughput of Ken depend on the underlying storage type (disk, no sync, and ramfs). The Bamboo-DHT results show MaceKen has 100% data resiliency under correlated failures and rolling restarts, which can happen in managed networks.

Ken and MaceKen are available online: http://ai.eecs.umich.edu/~tpkelly/Ken and http://www.macesystems.org/maceken.

Todd Tannenbaum, University of Wisconsin-Madison, asked if Ken messages have leases. He said that when resends are hidden, applications may not want to wait forever. For example, a seller may not want to wait forever for payment. Sunghwan answered that each event works as a transaction, so events would not lead to incorrect states. Todd again asked how this applies to waiting forever. Sunghwan said that Ken provides fault tolerance for crash-restart failures. Timeouts could be implemented at a higher layer. Someone asked whether Ken can roll back or cancel a transaction. Sunghwan answered that Ken can recover to the latest checkpoint.

Multicore

Summarized by Wonho Kim (wonhokim@cs.princeton.edu)

Managing Large Graphs on Multi-Cores with Graph Awareness

Vijayan Prabhakaran, Ming Wu, Xuetian Weng, Frank McSherry, Lidong Zhou, and Maya Haridasan, Microsoft Research

Vijayan Prabhakaran from MSR presented Grace, an in-memory transactional graph management systems, which can efficiently process large-scale graph-structured data by utilizing multicores in machines.

To exploit multicore parallelism, Grace partitions a given graph into smaller subgraphs that can be processed by each core separately, combines the results at a synchronization barrier, and continues the iteration. Vijayan mentioned that many graph algorithms, such as Google's page-rank, will work in this manner. In addition to the graph-specific optimizations, another interesting feature of Grace is supporting transactions by creating read-only snapshots.

In the evaluation, he compared the performances of different graph partitions. As expected, careful vertex partition leads to better performance than random algorithm. However, it was interesting that the vertex partitions do not make a difference when the number of partitions is low because (1) the partitions fit within a single chip and (2) the communication cost between partitions is very low in this case. Rearranging vertexes also improves performance by exploiting vertex locality in each partition. However, dynamic load-balancing among partitions does not improve overall performance.

Alexandra Fedorova, Simon Fraser University, asked about creating well-balanced partitions (static) and load balancing (dynamic). Grace adjusts the vertexes among the graph partitions at runtime to improve overall completion time. Vijayan answered that dynamic load-balancing is still needed because processing time in each partition is affected by multiple factors depending on the algorithms used.

MemProf: A Memory Profiler for NUMA Multicore Systems

Renaud Lachaize, UJF; Baptiste Lepers, CNRS; Vivien Quéma, GrenobleINP

Baptiste Lepers from CNRS presented MemProf, a memory profiler for NUMA systems that enables application-specific memory optimizations by pointing out the causes of remote memory accesses.

In NUMA systems, remote memory accesses have lower bandwidth and higher latency than accesses within the same node. The talk started with showing that many existing systems suffer from inefficient remote memory accesses in their default settings, and that NUMA optimizations can significantly improve their performance. However, existing profiles do not point out the causes of remote accesses, which is needed for making optimization decisions.

MemProf provides information about thread-object interactions in a given program from the viewpoints of both objects and threads. This output is useful for identifying what kinds of memory optimizations are needed. An interesting example presented in the talk is that the authors significantly improved the performance of FaceRec (face-recognition program) by more than 40% simply by replicating a matrix that is remotely accessed. MemProf tracks the object/thread life cycle using kernel hooks, but the overhead is quite low (5%).

Haibo Chen asked if replicating memory objects could increase cache misses. Baptiste answered that such an effect was not visible in the experiments and that the replication helps reduce remote accesses in a program. Baptiste also mentioned that, from MemProf output, users can detect different latencies among multiple nodes in NUMA systems. A follow-up question was about how MemProf can replicate memory objects automatically. Baptiste said that MemProf users should know the memory usage in the program because replication is possible only when memory is not fully utilized. He also mentioned that it would be difficult to optimize a program if the program exhibited different memory access patterns across executions. He also explained memory access patterns in Apache.

Remote Core Locking: Migrating Critical-Section Execution to Improve the Performance of Multithreaded Applications

Jean-Pierre Lozi, Florian David, Gaël Thomas, Julia Lawall, and Gilles Muller, LIP6/INRIA

Jean-Pierre Lozi started by showing that memcached performance collapses in manycore systems because lock acquisition time in critical sections increases as more cores are used. To address the lock contention cost, Remote Core Locking (RCL) executes highly contended critical sections in a dedicated server core, which removes atomic instructions and reduces cache misses for accessing shared resources. RCL requires dedicated cores, so it profiles applications to find candidate locks for RCL.

In micro-benchmarks, RCL shows much lower execution time compared to spin-lock, POSIX, and MCS. It was interesting to see that RCL improves the performance even in low-contention settings because execution in a dedicated core improves locality. RCL also significantly improves the performance of existing systems including memcached.

John Griffin from Telecommunication Systems asked what was the CPU utilization in the server cores during benchmarks. Jean-Pierre answered that the server cores are never idle; they always check pending critical sections to execute. Xiang Song from Shanghai University asked how RCL handles nested locks. RCL puts them in the same server core.

Packet Processing

Summarized by Wonho Kim (wonhokim@cs.princeton.edu)

The Click2NetFPGA Toolchain

Teemu Rinta-aho and Mika Karlstedt, NomadicLab, Ericsson Research; Madhav P. Desai, Indian Institute of Technology (Bombay)

Teemu Rinta-aho presented Click2NetFPGA, a compiler toolchain that automatically transforms software (in C++) to functional target hardware design (in NetFPGA).

Although many HLS tools are available, they are not made for people who do not understand hardware. Click2NetFPGA does not require knowledge in target hardware systems, and converts a given Click module to NetFPGA design. The talk mainly focused on the prototype implementation. In Click2NetFPGA, Click modules and configurations are first compiled into LLVM IR, and transformed to VHDL (VHSIC hardware description language) modules using AHIR compile developed from IIT Bombay.

The measurement results showed that Click2NetFPGA can reach only 1/3 of the line speed (1 Gbps) because of the inefficient translation between NetFPGA and Click data models. The presenter introduced their ongoing work on improving the performance of resulting hardware.

In the Q&A session, there was a question about how fast the compiled NetFPGA module is compared to the original Click software. Teemu answered that it could easily get 1 Gbps on a standard PC. Eddie Kohler from Harvard University asked what mistakes would be made if people work on similar projects. Teemu said " carefully study the source systems," which was an interesting answer because Eddie Kohler was the person who wrote the source system, the Click Modular Router.

Building a Power-Proportional Software Router

Luca Niccolini, University of Pisa; Gianluca Iannaccone, RedBow Labs; Sylvia Ratnasamy, University of California, Berkeley; Jaideep Chandrashekar, Technicolor Labs; Luigi Rizzo, University of Pisa and University of California, Berkeley

Luca Niccolini presented a software router that achieves energy efficiency by consuming power in proportion to incoming rates with a modest increase in latency.

While network devices are typically underutilized, the devices are provisioned for peak load. However, the devices are power-inefficient and consume 80–90% of maximum power, even with no traffic. Luca showed that CPU is the biggest power consumer in software routers. The authors developed an x86-based software router that adjusts the number of active cores and operating frequency based on incoming rate to improve energy efficiency. The design of the power control algorithm is guided by measurement of power consumption in different settings. It was interesting to see that running a smaller number of cores at higher frequency is more energy-efficient than running more cores at lower frequency.

In the evaluation, Luca showed that the new router consumes power in proportion to the input workload when running IPv4 routing, IPSec, and WAN optimization, saving 50% power. The tradeoff is latency, but it is a modest increase (10 μs). Another promising result was that the router did not incur packet loss or reordering in the experiments.

Someone asked if manipulating packet forwarding tables can overload some other cores. Luca answered that it is possible but the controller could detect such an event and change configuration. Luca also pointed out that reordering did not occur, because queue wakeup latency prevented packets in an empty queue from forwarding earlier than the other packets. Herbert Bos from Vrije University asked about an alternative approach, running different applications to different cores at different frequencies. This was not considered in the work, however.

netmap: A Novel Framework for Fast Packet I/O

Luigi Rizzo, Università di Pisa, Italy

▸ *Awarded Best Paper!*

Luigi Rizzo explored several options for direct packet I/O such as socket, memory-mapped buffers, running within the kernel, and custom libraries. But these all have issues with performance, safety, and flexibility. From measurement of packet processing time in different levels, Luigi showed that the three main costs come from dynamic memory allocation, system calls, and memory copies. netmap uses preallocated and shared buffers to reduce the cost.

netmap can transmit at line rate on 10 Gbps interfaces while receive throughput is sensitive to packet size because of hardware limitations in the system (e.g., cache line). netmap also improved the forwarding performance of Openvswitch and Click by modifying them to use netmap. It was interesting to see that netmap-Click in userspace can outperform the original Click running in the kernel.

Monia Ghobadi from the University of Toronto asked about inter-arrival times of back-to-back packets. Luigi said that packets were generated with no specified rate in the experiments.

Toward Efficient Querying of Compressed Network Payloads

Teryl Taylor, UNC Chapel Hill; Scott E. Coull, RedJack; Fabian Monrose, UNC Chapel Hill; John McHugh, RedJack

Teryl Taylor from UNC Chapel Hill presented an interactive query system, which can be used for forensic analysis. It is challenging to build an interactive query system for network traffic because network traffic typically has extremely large volumes, multiple attributes, and heterogeneous payloads. Teryl presented a solution which was to build a low I/O bandwidth storage and query framework by reducing, indexing, partitioning data and allowing application-specific data schemas.

In the evaluation, the authors used two data sets: campus DNS data and campus DNS/HTTP. The query system significantly reduced query processing time to sub-minute compared to PostgreSQL and SiLK for different query types (heavy hitters, partition intensive, and needle in a haystack).

There was a question about configuring on the fly what to store about the payload. Teryl answered that it is possible to create/install different versions of payloads. Keith Winstein from MIT asked how difficult it is to write a program that finds interesting patterns about a given suspect trace. Teryl said that using the interactive query system makes a huge difference in finding traffic patterns.

Plenary

Summarized by Rik Farrow (rik@usenix.org)

Build a Linux-Based Mobile Robotics Platform (for Less than $500)

Mark Woodward, Actifio

Mark Woodward told us that he has worked for robotics companies for many years, starting with Denning Mobile Robotics in 1985. But by then, he had already built his own robot, based on a Milton Bradley Bigtrak chassis, a programmable tank from 1979. The Bigtrak had a simple Texas Instruments microcontroller, and Mark used this to lead into a discussion of CPUs that appeared in later robots, such as Motorola 68K and Z80 CPUs, as well as sensors.

Mark wasn't very excited about the state of commercial robotics. He called the Roomba a "Bigtrak with a broom," the Segway as a great example of process control, and self-parking cars as something that sometimes works. He described a project he had worked on while at Denning: a robotic security guard. When they tried to sell their 1985 $400,000 robot, they discovered that watchguard companies preferred to hire guards for a lot less. The robot itself was so expensive, thieves might elect to steal it and ignore what the robot was guarding.

Mark had brought his own robot with him, and he explained the technology he used to build it. For example, it uses a smaller form factor motherboard (ITX) for general processing, for connection to video cameras, and for running text-to-speech processing, so the robot can talk via speakers connected to the motherboard. While the motherboard runs Linux, Mark prefers to use an Arduino for sensors and for motor control. He explained that the motor control was actually very difficult, as simply measuring how much each wheel turns doesn't actually reflect the movement of the robot, as wheels can (and do) slip on many surfaces. The motor control uses a Proportional-Integral-Derivative (PID) algorithm, a commonly used feedback controller.

Mark then provided a list of tools useful for building robots and other hardware products: temperature controlled soldering iron, oscilloscope (Rigil), benchtop power supplies, as well as more mundane items like lawnmower wheels, duct tape, tap and dies, wire ties, and PVC pipe. He also recommended the book *The Art of Electronics* (Paul Horowitz, Winfield Hill), but Clem Cole (Intel) countered that *Practical Electronics for Inventors* (Paul Scherz) is a better and more recent book.

Bill Cheswick (Independent) asked how Mark dealt with surface mounts, and Mark answered that he didn't use them. Clem mentioned that there are workarounds for flowing solder to attach surface mounts. Steve Byar (NetApp) asked about power supplies, and Mark suggested contacting him later. Rik Farrow asked whether he had considered using an optical mouse as a sensor to gain movement information, and Mark said he hadn't, but didn't think it would work. Clem Cole asked about using stepper motors, and Mark described them as "evil," requiring a separate input for each step. Marc Chiarini (Harvard SEAS) asked about making robots like Mark's smaller. Mark pointed out that his robot had an extra large, Plexiglas top that he used for scaffolding, to hold things like speakers and video cameras which could be removed. Marc then asked about the size of the wheels. Mark replied that he is using plastic gearing, so the wheels need to have a large diameter. Ches asked what tools did Mark wish he'd had when he started. Mark said an oscilloscope.

Security

Summarized by Tunji Ruwase (oor@cs.cmu.edu)

Body Armor for Binaries: Preventing Buffer Overflows Without Recompilation

Asia Slowinska, Vrije Universiteit Amsterdam; Traian Stancescu, Google, Inc.; Herbert Bos, Vrije Universiteit Amsterdam

Asia Slowinska presented a tool called BinArmor, that hardens C binaries, even without symbol information, against buffer overflow attacks against both control-data and non control-data. The work was prompted by statistics that show that despite its buffer overflow vulnerabilities, C still remains the most popular programming language. Moreover, current techniques are ineffective for protecting binaries (e.g., legacy code) against buffer overflow attacks. By detecting non-control data attacks, BinArmor provides better protection than taint analysis, which only detects control data attacks. However, BinArmor is prone to false negatives due to its reliance on profiling (as discussed later); i.e., it can miss real attacks.

To harden a program binary against attacks, BinArmor (1) finds the arrays in the program, (2) finds the array accesses, and (3) rewrites the binary, with a novel color tracking code, for buffer overflow detection. The Howard reverse engineering tool (presented at NDSS 2011) is used to detect arrays in binaries without symbol information. Next, profiling runs of the program are used to detect accesses to the detected arrays. Coverage issues of profiling lead to the false negatives in BinArmor. The binary rewrite step assigns matching colors to each pointer and the buffer it references, tracks color propagation, and checks that the color of de-referenced pointers matches the referenced buffer. Protecting fields (and subfields) of C structs requires a more complex coloring scheme, where fields have multiple colors, to permit the field to also be accessed through pointers to the enclosing structs.

Asia then presented the evaluation of BinArmor, which focused on bug detection effectiveness and performance. BinArmor detected buffer overflows in real world applications, including a previously unknown overflow in the htget program. Also, it introduced, at most, a 2x slowdown in real world I/O-intensive programs. The nbench benchmark suite, which is more compute intensive, had a worst case slowdown of 5x, with a 2.7x average slowdown.

A lively question/answer session ensued, with a session-leading number (six) of questioners. Bill Cheswick set the ball rolling by asking if BinArmor detected new bugs; Asia referred to the htget overflow. Andreas Haeberlen from University of Pennsylvania asked how an attacker could adapt to BinArmor. Asia pointed out that the coverage issues of the profiling step could be exploited. Larry Stewart asked how pointers used by memcpy (and other libc functions) were handled by BinArmor, since these pointers travel through many software layers. Asia responded that more pointer tracking would be required for that. Julia Lawall asked if BinArmor currently performed any optimizations, and suggested bounds-checking optimizations in Java. Asia responded that optimizations were future work. Konstantin Serebryany from Google asked if Body Amour reported errors for libc functions that read a few bytes beyond the buffer. Asia clarified that this was not a problem in practice, because the granularity of colors in BinArmor is 4 bytes. Steffen Plotner of Amherst College asked if BinArmor could be used to protect the Linux kernel. Asia responded that they had not tried.

Abstractions for Usable Information Flow Control in Aeolus

Winnie Cheng, IBM Research; Dan R.K. Ports and David Schultz, MIT CSAIL; Victoria Popic, Stanford; Aaron Blankstein, Princeton; James Cowling and Dorothy Curtis, MIT CSAIL; Liuba Shrira, Brandeis; Barbara Liskov, MIT CSAIL

Confidential data, such as credit card information, and medical records, are increasingly stored online. Unfortunately, distributed applications that manage such data, are often vulnerable to security attacks, resulting in high profile data theft. Dan Ports introduced the Aeolus security model, which uses decentralized information flow control (DIFC), to secure distributed applications against data leaks. Dan observed that access control was not flexible enough for this purpose, because the objective is to restrict the use, not the access, of information. Aeolus describes a graph-based security model and programming abstractions for building secure distributed applications.

In summary, Aeolus tracks information flow within a protection boundary to ensure that only declassified information flows outside the boundary. Aeolus achieves this using three concepts: *principals* (entities with security concerns, e.g., individuals), *tags* (the security requirements of data), and *labels* (set of tags). Labels associated with data objects are immutable, while threads are associated with principals and mutable labels (reflecting accessed data). Aeolus also maintains an authority graph, to ensure that declassification (tag removal) is done by authorized principals. Dan further discussed Aeolus programming abstractions and Java implementation.

Evaluations using micro-benchmarks showed that most Aeolus operations are within an order of magnitude of Java method calls. Moreover, Aeolus imposed a mere 0.15% overhead on a financial management service application. The low overhead is because the Aeolus operations are infrequent and relatively inexpensive. Aeolus is available at http://pmg.csail.mit.edu/aeolus.

Someone expressed concern about malicious information flowing into the system. Dan confirmed that Aeolus in fact tracks the integrity of incoming information, and referred the audience to the paper for details. Rik Farrow also expressed a concern that conventional uses of authority graphs, e.g., in banks, often suffered from untimely updates. Dan observed that untimely updates were due to centralized control, thus decentralization in Aeolus helped to avoid the problem.

TreeHouse: JavaScript Sandboxes to Help Web Developers Help Themselves

Lon Ingram, The University of Texas at Austin and Waterfall Mobile; Michael Walfish, The University of Texas at Austin

Third-party code is extensively used by JavaScript applications and, allowed to execute with similar privileges, is therefore trusted to be safe/correct. Lon Ingram demonstrated this was misplaced trust. For example, using a third-party widget that had a hyperlink vulnerability for processing online payments, he showed how this vulnerability could be exploited by an attacker to steal credit card information. He then presented TreeHouse, a system that uses sandboxing to enable safe use of third-party code in JavaScript applications.

TreeHouse is implemented in JavaScript, and is therefore immediately deployable, as no browser changes are required. Moreover, it modifies the Web Worker feature of modern browsers to act as containers for running third-party code. By transparently interposing on privileged operations, TreeHouse enables flexible control of third-party code. Lon then showed how an application can use TreeHouse to implement the required security policies for thwarting the attack in the motivating example.

Experimental results showed that Document Object Model (DOM) use significantly affected TreeHouse overheads. In particular, DOM access can be up to 120k times slower with TreeHouse. Also, TreeHouse increases initial page load latency by 132–350 ms, on average. Consequently, TreeHouse is not suitable for DOM-bound applications or applications with a tight load time. Further information about TreeHouse is available at github.com/lawnsea/Treehouse and lawnsea@gmail.com.

James Mickens from MSR asked whether the prototype chain needed to be protected. Lon said that he would have to think about it. Konstantin Serebryany from Google asked what JavaScript feature would Lon like to change. Lon responded that he would like parent code to run child code with restricted global symbol access. Steve McCaant from UC Berkeley highlighted a regular expression typo in the slide, which Lon acknowledged.

Cloud Terminal: Secure Access to Sensitive Applications from Untrusted Systems

Lorenzo Martignoni, University of California, Berkeley; Pongsin Poosankam, University of California, Berkeley, and Carnegie Mellon University; Matei Zaharia, University of California, Berkeley; Jun Han, Carnegie Mellon University; Stephen McCamant, Dawn Song, and Vern Paxson, University of California, Berkeley; Adrian Perrig, Carnegie Mellon University; Scott Shenker and Ion Stoica, University of California, Berkeley

Stephen McCamant presented Cloud Terminal, a system for protecting sensitive information on PCs. Cloud Terminal assumes that the vulnerabilities in client software stack, including the OS, can compromise the confidentiality and integrity guarantees offered by prior techniques. Therefore, Cloud Terminal proposes a new software architecture for secure applications running on untrusted PCs, with a Secure Thin Terminal (STT) running on client systems, and remote applications in a Cloud Rendering Engine (CRE) VM. As an example, Stephen demonstrated how Cloud Terminal allows a PC user to perform secure online banking without any dependence on the untrusted OS.

On the client system, STT's role is to render graphical data from the remote application, and forward keyboard and mouse events to it. A simple hypervisor, called Microvisor, leverages Flicker and Intel TXT to isolate STT from the client OS. STT was implemented in 21.9 KLOC. CRE runs the remote application in a VM, and connects to STT via a lightweight remote frame buffer VNC protocol with SSL security. CRE incorporates a number of techniques to provide scalability (support for 100s of application VMs) and security.

Cloud Terminal was evaluated with CRE running on a 2 GHz, 16-core system, with 64 GB RAM, while STT ran on a Lenovo W510 laptop. The evaluated applications were AbiWord, Evince, Wells Fargo online banking on Firefox, and Gmail on Firefox. The applications were found to be quite usable, with reasonable display and latency. However, page scrolling was sluggish, but Stephen said that this could be optimized. In terms of cost, Cloud Terminal could provide secure computing services at 5 cents per user per month.

Andreas Haeberlen, University of Pennsylvania, asked if client-side resources could be used to improve performance. Stephen replied that, while this was possible, it would not match the target applications. Someone asked if more client-side devices could be supported in STT. Stephen said supporting the drivers would increase complexity and thus undermine trustworthiness.

Short Papers: Tools and Networking

Summarized by Rik Farrow (rik@usenix.org)

Mosh: An Interactive Remote Shell for Mobile Clients

Keith Winstein and Hari Balakrishnan, MIT Computer Science and Artificial Intelligence Laboratory

Keith Winstein gave a lively talk about a mobile shell, Mosh. Keith began by saying that everyone uses SSH, but SSH uses the wrong abstraction: an octet stream. What you want when you use SSH is the most recent state of your screen. He joked that today's network is not like the ARPANET, which was much faster. The authors developed SSP, the state synchronization protocol, which communicates the differences between the server's concept of a screen and the screen at the client side. Mosh also displays keystrokes, as well as backspace and line kill, immediately, on the user's terminal, underlining characters until the server confirms any local updates.

Mosh still uses SSH to authenticate and start up a mosh_server. When mosh_server starts up, it communicates an AES key over SSH before shutting down that connection. Mosh_client uses that key in aes-ocb mode, which supplies both encryption and an authenticated stream. Neither the mosh_server or client run with privileges. Mosh uses UDP packets, which means that there is no TCP connection to maintain. Using UDP with AES-OCB (AES Offset Codebook mode) is what allows the Mosh user to roam. Mosh also manages its own flow control that adapts to network conditions.

Keith finished with a demo comparing SSH and Mosh. When the IP address changes, SSH doesn't even tell us that the connection is dead, Keith said, and that is "most offensive."

Lois Bennett asked about configuring a firewall to allow Mosh, and Keith replied that you need to keep a range of UDP ports open, depending on how many simultaneous Mosh sessions you expect. He also said they are working to make Mosh more firewall friendly. Someone else wondered how Mosh could behave predictively with Gmail, and Keith responded that Gmail is actually easier to handle than terminal applications like Emacs.

The August 2012 issue of *;login:* includes an article about Mosh.

TROPIC: Transactional Resource Orchestration Platform in the Cloud

Changbin Liu, University of Pennsylvania; Yun Mao, Xu Chen, and Mary F. Fernández, AT&T Labs—Research; Boon Thau Loo, University of Pennsylvania; Jacobus E. Van der Merwe, AT&T Labs—Research

Changbin Liu described a problem with how IaaS cloud providers provision services: if one link in a chain of events fails, the entire transaction fails. For example, starting a server requires acquiring an IP address, cloning the OS image within storage, creating the configuration, and starting the VM. The key idea behind TROPIC is that it orchestrates transactions with ACID for robustness, durability, and safety. TROPIC has a logical layer with a replicated datastore that communicates with the physical data model. TROPIC runs multiple controllers with a leader and followers. If a step fails, TROPIC rolls back to the previous stage, and continues with the failure hidden from the user. TROPIC also performs logical layer simulations to check for constraint violations—for example, allocating more memory than the VM host has, or using the next hop router as a backup router, the very problem that caused the failure of EC2 in April 2011.

They have an 11k LOC Python implementation which they have tested on a mini-Amazon setup deployed on 18 hosts in three datacenters. The code is open source, and will be integrated into Open Stack.

Haibo Chen (Shanghai Jiao Tong University) asked how they can tell the difference between a true error and excess latency. Changin Liu replied that error detection is via error message. If the connection hangs for a minute, TROPIC kills the connection or terminates it. There is more about error handling in the paper.

Trickle: Rate Limiting YouTube Video Streaming

Monia Ghobadi, University of Toronto; Yuchung Cheng, Ankur Jain, and Matt Mathis, Google

Monia Ghobadi explained that the way videos are streamed by YouTube and Netflix results in bursts of TCP traffic. The bursty nature of this traffic causes packet losses and affects router queues. YouTube writes the first 30–40 seconds of a video, followed by 64 KB blocks using a token bucket to establish a schedule. Netflix sends out 2-MB bursts, also causing periodic spikes. Trickle uses the congestion window to rate limit TCP on the server side. Trickle requires changes to the server application. Linux already allows setting a per-route option called cwnd_clamp, and they wrote a small kernel patch to make this option available for each socket.

Monia compared the current YouTube server, ustreamer, with Trickle, using data collected over a 15-day period in four experiments in Europe and India. Trickle reduced retransmissions by 43%. Sending data more slowly also affects queueing delay, with roundtrip times (RTTs) lower than ustreamer by 28%. She then demonstrated a side-by-side comparison of ustreamer and Trickle (http://www.cs.toronto.edu/~monia/tcptrickle.html) by downloading movie trailers. In the demo, Trickle actually worked faster, slowly moving ahead of the display in the ustreamer window because of ustreamer packet losses.

Someone from Stanford asked if the connection goes back to slow start when the connection is idle. Monia answered that since they are using the same connection, the congestion window clamp still exists. John Griffinwood (Telecom Communications) wondered whether they saw jitter and whether Google had adopted Trickle. Monia answered that Trickle dynamically sets the upped bound and readjusts the clamp if congestion is encountered. While she was working as an intern for Google, they had planned to implement Trickle. Someone from AT&T asked whether mobile users also benefit from this. Monia answered yes.

Tolerating Overload Attacks Against Packet Capturing Systems

Antonis Papadogiannakis, FORTH-ICS; Michalis Polychronakis, Columbia University; Evangelos P. Markatos, FORTH-ICS

Antonis Papadogiannakis told us that when a packet capture system gets overloaded, it randomly drops packets. When a system is being used for intrusion detection, random drops are not good, as the dropped packets may be important. An attacker could even cause the overload by sending packets that result in orders of magnitude slower processing, or using a simpler but more direct DoS attack. Antonis pointed out that existing solutions include over-provisioning, thresholds, algorithmic solutions, selective discarding, and ones that attempt to reduce the difference between average and worst case performance.

Their solution is to store packets until they can be processed. Excess packets are buffered to secondary storage if they don't fit in memory, so all packets will be analyzed. When the ring buffer gets full, packets are written to disk. When the ring buffer has space again, packets are read back and processed. If the system running packet capture is also relaying packets, this will result in additional latency. But this may not be an unreasonable price to pay if you are relying on this system to block attacks.

The limitation to their approach are the delays when relaying and the practical limitation of buffering packets to disk. They tested their implementation using a modified version of libpcap evaluated with Snort, using an algorithmic complexity attack which resulted in an unmodified system losing as much as 80% of packets at one million packets per second. Their system did not lose any packets at this rate. There were no questions.

Enforcing Murphy's Law for Advance Identification of Run-Time Failures

Zach Miller, Todd Tannenbaum, and Ben Liblit, University of Wisconsin—Madison

Zach Miller explained that Murphy causes "bad things" to happen to the software under test. Using ptrace, Murphy captures all system calls and modifies the returned results. Murphy follows POSIX behavior when generating responses, so the results should not be that far afield from things that an application might be expected to handle properly, such as a failed write() system call because of a disk full error. Murphy works with any language, is done in user space, and tests entire software stacks, since it interposes on system calls going to the kernel.

Murphy found a bug in /bin/true, because the command expects read() to succeed. Murphy includes rich constraints, such as regex matching, state, mapping file descriptors to filenames, and other tricks. Murphy can simulate full disks, time going backwards, and other results that are allowed by system calls. Murphy keeps a log of all changes it made, and this log can be replayed to test fixed code. Murphy can also skip through the replay log and suspend the application right before the return result that caused a crash.

They found bugs in C, Perl, Python, and OpenSSL in their testing. At this point, Murphy only works under 64-bit Linux.

Eddie Kohler (Harvard) wondered, if they find a bug under Linux, is it a true bug in other environments? Zach said that because Linux is a POSIX-compliant system, bugs found there will be true for any POSIX-compliant software. Alexander Potapenko (Google) asked about the performance overhead. Zach responded that it varied based on the amount of system calls made by the application under test. It might be as little as six times slower, and as much as 60 times.

Distributed Systems

Summarized by Brian Cho (bcho2@illinois.edu)

A Scalable Server for 3D Metaverses

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

Ewen Cheslack-Postava explained that a Metaverse is a 3D space, where everything in the space is editable by users. There are a wide variety of applications, including games, augmented reality, etc. Unfortunately, what you get today is not as pretty as artist renderings. Examples of artist renderings were shown, followed by a very spare screen from Second Life. The reason Second Life looked so spare was because the system won't display things more than a few meters away. A second screen, shown after the user moved a few steps shows a much richer world. The problem is that the system doesn't know how to scale, while not sacrificing user experience.

These are systems problems. The only way currently to scale is to carve the world geographically into separate servers, and limit each server to communication with a few neighboring servers. This work uses the insight that the real world scales, and scales by applying real-world constraints to the system. Because there is a limited display resolution, they use a technique called solid-angle queries. The solid angle dictates how large an object appears, and anything with a large solid angle should show up. So, for example, mountains should show up, even if they are far away. The second thing done is to combine objects. The combination of both solid-angle queries and aggregates is close to ideal.

These techniques are used through a core data structure called Largest Bounding Volume Hierarchy (LBVH) tree structure, which modifies the Bounding Volume Hierarchy (BVH) tree. An example of four objects, in a three-level hierarchy was shown. BVH uses spheres that can contain objects, and hierarchically combines neighboring spheres into ever-larger spheres. The problem with this structure, is that to find large objects to display, a long recursive search has to be done, and because the spheres overestimate size, it's hard to prune parts of the search. LBVH instead stores the largest object in a subtree at interior nodes. Doing this results in 75–90% fewer nodes tested. Other techniques are also presented, showing how to deal effectively with moving objects, and redundant queries. Aggregation is applied by storing an aggregated object of lower quality on each internal node (BVH only stores objects at the nodes). Queries on LBVH across different servers are done efficiently by running large queries across machines, and then filtering those for each individual query.

An example application, Wiki World, was shown. You can automatically find info about objects on Wikipedia. This would not be possible in other systems. Many more systems challenges at the intersection of systems, graphics, PL, databases, etc. are present in this area. An example is audio: for instance, playing a distant siren or the combined roar of a crowd. More info can be found at http://sirikata.com.

Jon Howell, Microsoft Research, asked what workload was used to measure the improvements. Ewen said it is hard to collect or generate workloads. What they used were a synthetic random workload, and a workload collected from Second Life. For their experiments, they tried both workloads. Chip Killan, Purdue, asked how direct communication is done with aggregate objects. Ewen said that you can't do this with aggregate objects currently, which is a limitation in the current system.

Granola: Low-Overhead Distributed Transaction Coordination

James Cowling and Barbara Liskov, MIT CSAIL

James Cowling told us that Granola is an infrastructure for building distributed storage applications. It provides strong consistency without locking for multiple repositories and clients. The unit for an atomic operation chosen is transactions. Why? Because using transactions allows concurrency on a single repository to be ignored. Transactions are allowed to span multiple repositories, avoiding inconsistency between repositories. However, distributed transactions are hard. Opting for consistency, e.g., using two-phase commit, results in a high transaction cost. Opting for performance, e.g., providing a weak consistency model, places the burden of consistency on application developers, which evidence suggests makes their job difficult.

To allow strong consistency and high performance, for at least a large class of transactions, this work provides a new transaction model. There are three classes of operations—first, those that work on a single repository, and then, for distributed operations, coordinated and independent transactions. Granola specifically optimizes for single and distributed independent operations; it provides one-round transactions. An example of a distributed independent operation was shown: consider transferring $50 between two accounts. Each participant must make the same commit/abort decision. Evidence shows this class of operations is common in OLTP workloads. For example, TPC-C can be expressed entirely using single or independent transactions.

Granola provides both a client library and a repository library, and sits between the clients and repositories. Each repository is in fact a replicated state machine. There are two modes for a repository—it will primarily be in timestamp mode, and occasionally switch to locking mode, when coordinated transactions are required. Each transaction is assigned a timestamp, and transactions are executed in timestamp order. Thus, timestamps define a global order. The challenge is how to assign timestamps in a scalable, fault-tolerant way.

For a single repository transaction, the steps of operation are: (1) the repository assigns a timestamp, chosen to be higher than any previous timestamps; (2) the transaction with the timestamp is logged; and (3) the transaction is executed. For distributed, independent transactions, repositories additionally vote to determine the highest timestamp. The steps are (1) propose, (2) log, (3) vote, (4) pick, and (5) run. The transaction will not execute until it has the lowest timestamp of all concurrent transactions. This guarantees a global serialized execution order. Granola provides interoperability with coordinated transactions, by requiring repositories to switch to lock mode. Locking is required to ensure a vote is not invalidated. The protocol is changed to include a preparation phase, and the transaction is aborted if there is a conflict. The repository can commit transactions out of timestamp order. The result will still match the serialized order, even if execution happens out of timestamp order (because of the nature of transactions). Repositories throughout the system can be in different modes.

Experiments were presented using the TPC-C benchmark. Granola scales well. With a higher load of distributed transactions, Granola throughput only goes down to half. This is because there is no locking or undo logging.

Marcos Aguilera, Microsoft Research, commented about the ambiguity of the terminology for consistency, that it could mean either serializability or atomicity. James agreed that database and system communities use different terminology. Marcos then asked if a change doesn't touch the entire repository, if there is a need to switch the entire repository to lock mode. James answered that if there were a separate object model, this would be possible, but in the system the application is just considered a blob, so it is not possible currently.

Timothy Zhu, CMU, asked for suggestions on when to use this system. Is it applicable all the time? James said there are obvious limitations; when there are failures, you have to switch into locking mode, so when you really only want availability, this isn't a great system. Timothy asked if the timestamps are similar to Lamport clocks. James answered that they are basically Lamport clocks, except that voting does not take place in Lamport clocks. Also, Granola in fact makes use of local system clocks at clients for performance.

Zhiwu Xie, Virginia Tech, asked James to compare Granola with the Calvin system. James answered that Calvin has an agreement layer that needs all-to-all communication, so they have higher latency. He believes there is a potential scalability limit because of this, but they showed 100 nodes, which is impressive. Calvin's advantage is that it has more freedom to shift transactions around. Granola is constrained, so it relies on single-threaded execution.

High-Performance Vehicular Connectivity with Opportunistic Erasure Coding

Ratul Mahajan, Jitendra Padhye, Sharad Agarwal, and Brian Zill, Microsoft Research

Ratul Mahajan started by asking how many of the audience have used Internet access on-board a vehicle. There was quite a show of hands. Riders love Internet access—it boosts ridership. But performance can be poor, and service providers don't have a good grasp on how to improve it. A service provider's support suggested, for example, that the user cancel a slow download and retry in approximately five minutes.

Vehicular connectivity uses WWAN links. It's not the WiFi that is bad, but rather that the WWAN connectivity is lossy. This is not due to congestion but is just how wireless behaves. Two methods to mask losses are retransmission and erasure coding (EC). Retransmissions are not suitable for high delay paths. So high-delay should use erasure coding. Existing EC methods are capacity-oblivious, meaning there is a fixed amount of redundancy. The problem is that this fixed amount may be too little or too much, relative to the available capacity. Thus, the main proposal is opportunistic erasure coding (OEC)—this uses spare capacity. The challenge is how to adapt given highly bursty traffic. Real data from MS commuter buses shows that you would have to adapt at very small time-scales.

The transmission strategy for OEC is to send EC packets if and only if the bottleneck queue is empty. This matches "instantaneous" spare capacity and produces no delay for data packets. As for the encoding strategy, conventional codes are not appropriate. These codes don't provide graceful degradation when the amount of redundancy provided is different from that needed. Thus, OEC is designed with greedy encoding. The strategy is that, if the receiver has a lot of packets, then EC has a lot of packets. A good property that is achieved is that each packet transmission greedily maximizes goodput.

PluriBus is OEC applied to moving vehicles. OEC happens between the VanProxy (on the moving bus) and LanProxy (part of the immobile infrastructure). Details of how relevant parameters are estimated were given. Ratul claimed that the aggressive use of spare capacity is not such a bad idea. The observation is that the network is not busy all the time using timeouts, and this means that network traffic only increases by a factor of two. Media access protocols isolate users from each other, so this won't hurt innocent users.

Evaluation was done through a deployment on two buses on the MS campus, with trace-driven workloads, and emulation. The main result is that performance is improved by 4x. The workload was scaled, up to a factor of 8x, to show that losing spare capacity is not a major concern. In emulation, it was shown that OEC outperforms other loss recovery methods. This is because retransmission requires delay, and fixed redundancy ECs are not opportunistic.

Philip Levis, Stanford, asked whether fountain codes could be used instead. Ratul replied that a challenge in PluriBus is that r is dynamic, in addition to being estimated. With fountain codes, k of n packets must arrive, which could not be guaranteed.

Bradley Andrews, Google, asked whether any actual user feedback was collected. Ratul answered there were two main reasons that they did not. First, outsourcers who ran the actual commute buses didn't allow changes, so this couldn't be applied to those buses. Second, during the study, a large shift to smartphones meant that the demand for Internet access on these buses essentially disappeared. Bradley then asked whether there was collaboration with wireless carriers. Ratul explained that permission was not asked of wireless carriers before the study, but once the study was over, the results were shared with carriers.

Masoud Jafavi, USC, asked what the effect of a crowded area would be. Ratul replied that experiments were not done to quantify this, but the feeling is that any kind of damage will not be too large. Rather, the important questions to consider are: Do you or do you not have a dedicated channel to the cell provider? And how many users can get a channel, and how quickly? Ratul commented that PluriBus may hold on to the channel for about 200 ms longer, but compared to the release timeout of five seconds, this is a small fraction of the overall time.

Server-Assisted Latency Management for Wide-Area Distributed Systems

Wonho Kim, Princeton University; KyoungSoo Park, KAIST; Vivek S. Pai, Princeton University

Wonho Kim presented this work on one-to-many file transfer. This may sound like an old problem: e.g., CDN, P2P, Gossip approaches have been around for a while. But these typically focus on bandwidth efficiency or delivery odds. The focus in this work is on the metric of completion time. This requires different strategies. Some motivating use cases are: (1) configuration to remote nodes—e.g., in a CDN; (2) distributed monitoring—e.g., coordinating before measurement; and

(3) developers—e.g., a long develop-deploy cycle in PlanetLab can hurt productivity.

The system developed is LSync. It provides a simple folder sync interface. The lessons and contributions are: (1) existing systems are suboptimal mainly because they are not favorable when there are slow nodes; (2) completion time depends on the set of target nodes, so LSync selects the best set of nodes; (3) end-to-end transfer can be faster than an overlay, because of startup latency, so overlay is used only when appropriate; (4) overlay performance changes at short time scales, so transfers are adapted while they are taking place. Existing systems assume an open client population, so their main goals are maximum average performance, maximum aggregate throughput, etc. LSync focuses only on internal dissemination within a fixed client population. Thus it aims to minimize completion time. This time is dominated by slow nodes.

LSync uses server's spare bandwidth to assist slow nodes. The question is how to do this efficiently. First, look at node scheduling—either do fast first, or slow first. Intuitively, fast first is optimal for mean response time, while slow first gives preference to nodes that are expected to be slow. The results show that in fast first, slow nodes become a bottleneck at the end. Slow first starts slower but ends quicker. But not every scenario requires waiting for 100% sync. LSync allows the specification of a fraction of nodes, called the target sync ratio. LSync integrates node selection with the aforementioned scheduling.

Leveraging an overlay mesh is scalable, but needs to be careful about startup latency. For small files, only using end-to-end transfer (E2E) is faster than using an overlay. For large files, overlay is faster than E2E. So, LSync should adapt to the target ratio, file size, bandwidth, etc. The approach used is that LSync monitors the overlay's startup latency. It splits nodes into an overlay group and E2E group, depending on the overlay connection speed, and tries to match the completion time of both. To deal with overlay performance fluctuation, adaptive switching is used.

Evaluation was done on PlanetLab, using multiple CDNs, and compared against multiple systems. A dedicated origin server was used with 100 Mbps bandwidth. LSync improves over other systems, by choosing E2E vs. overlay rates. Less variation is shown with adaptive switching.

Jon Howell, Microsoft Research, asked for clarification of the target completion ratio—whether or not they care about which specific nodes are completed. Wonho answered that you can do both. You can simply tune the completion ratio if you aren't concerned which set of nodes are completed. If you do care, you can specify a new set of nodes, and then run with completion ratio set to 1.0.

Deduplication

Summarized by Anshul Gandhi (anshulg@cs.cmu.edu)

Generating Realistic Datasets for Deduplication Analysis

Vasily Tarasov and Amar Mudrankit, Stony Brook University; Will Buik, Harvey Mudd College; Philip Shilane, EMC Corporation; Geoff Kuenning, Harvey Mudd College; Erez Zadok, Stony Brook University

Deduplication is the process of eliminating duplicate data in a system and has been the focus of a lot of prior work. Unfortunately, most of the prior work has looked at different data sets, and so it is almost impossible to compare the performance of these different deduplication approaches. A survey of the data sets used by 33 deduplication papers was conducted by the authors and they found that most of the data sets were either private (53%), hard to find (14%), or contained less than 1 GB of data (17%). Thus, there is a need for an easily accessible data set with configurable parameters.

In order to create realistic data sets, Vasily Tarasov presented work to accurately track how file systems mutate over time. They do so by observing consecutive snapshots of real data sets, combined with a Markov model and multi-dimensional analysis. Comparison with the evolution of real file system images shows that the authors' emulation approach very accurately tracks the number of chunks and files over time, as well as the number of chunks with a given degree of duplication. Importantly, the file system profile sizes generated by the authors are 200,000 times smaller than the real profile sizes. The emulation time is proportional to the size of the data set, with a 4 TB data set emulation requiring about 50 minutes.

Haibo Chen from Shanghai Jiao Tong University asked about the differences in numbers between emulation and live file systems. Vasily answered that the emulation is a statistical process and so there would naturally be differences from time to time between emulation and the live system. However, Vasily felt that the emulation was close enough to the live system evolution. Haibo then asked whether the emulation runtime could be reduced by parallelization. Vasily agreed that it could; in their current work, scanning the data sets is done in parallel, but everything else is serialized, and thus, there is potential for parallelization.

An Empirical Study of Memory Sharing in Virtual Machines

Sean Barker, University of Massachusetts Amherst; Timothy Wood, The George Washington University; Prashant Shenoy and Ramesh Sitaraman, University of Massachusetts Amherst

Sean Barker presented this work which analyses the potential of page sharing in virtualized environments. Page sharing is a popular memory deduplication technique for virtual machines in which duplicate pages are eliminated. There has been a lot of prior work in exploiting page sharing for deduplication, with recent publications eliminating more than 90% of duplicate memory pages. However, the levels of sharing typically seen in real-world systems and the factors that affect this sharing remain open questions. The goal of the authors' work is to answer such questions.

The authors looked at a wide variety of memory traces, including uncontrolled real-world traces as well as controlled, configurable synthetic traces. Results indicate that sharing within a single VM (self-sharing) is about 14%, whereas sharing between VMs is only about 2%. Thus, 85% of the potential for deduplication is within a VM, indicating (very interestingly) that page deduplication is quite useful even for non-virtualized systems. Further investigation revealed that most of the self-sharing (94%) is because of shared libraries and heaps. However, the amount of self-sharing is largely impacted by the choice of base OS. Likewise, sharing across VMs is also impacted by the base OSes, with sharing being significant when the VMs have the same base OS as opposed to different base OSes.

The case study drew a lot of questions from the audience. Thomas Barr from Rice University asked whether the authors had looked at sharing larger multiples of page sizes. Sean answered that they didn't look much at coarse-grained sharing since the amount of sharing in this case was much smaller. Ardalan Kangarlou from NetApp asked whether the numbers for sharing in prior work were higher because they looked at synthetic workloads. Sean replied that the amount of sharing depends on the data set, and for the uncontrolled data set that he was looking at, the sharing was much lower. He urged the audience to look at actual data sets. Someone noted that memory contents change from time to time, and wondered whether vendors really benefit from sharing. Sean acknowledged that short-lived data is hard to capture, but that was a separate issue. Jiannan Ouyang from University of Pittsburgh asked about the size of memory footprint in the workload. Sean answered that the VMs they used had 2 GB of memory (each) on them, and since they had lots of applications running, he guessed that a good portion of the 2 GB memory was being used.

Primary Data Deduplication—Large Scale Study and System Design

Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Oltean, Jin Li, and Sudipta Sengupta, Microsoft Corporation

Sudipta Sengupta and Adi Oltean jointly presented this work, which will be part of Windows Server 2012. Sudipta started this presentation, which looks at deduplication in primary data, as opposed to the more common case of backup data. Primary data deduplication is important because of the continuing growth in the size of primary data and because this is the number one technology feature that customers are looking for when choosing a storage solution. The main challenge in primary data deduplication is that it needs to be non-intrusive to the workload.

The key design decision made by the authors is to post-process deduplication, which helps to schedule deduplication in the background when data is not hot. Also, the authors decided to use a larger chunk size (80 KB), which helps to reduce metadata, and thus reduces deduplication overhead. To compensate for the loss in deduplication opportunity due to larger chunk sizes, the authors use chunk compression. The authors also modify the basic fingerprint-based chunking algorithm to reduce the forced chunk boundaries at the maximum chunk size and to obtain a more uniform chunk size distribution. Lastly, in order to reduce the RAM footprint and the number of disk seeks, Adi presented the idea of partitioning the data, then performing deduplication on each partition, and, finally, reconciling the partitions by deduplicating across them. Performance evaluation of this approach reveals that deduplication throughput is about 25–30 MBps, which is about three orders of magnitude higher than previous work. Deduplication takes up 30–40% of one core, leaving enough room (assuming a manycore server) for serving primary workload.

Haibo Chen from Shanghai Jiao Tong University asked about the effects of data corruption on deduplication. Adi replied that they have looked at corruption and recovery, but this was not part of the paper. Essentially, they ensure that in case of a crash, data can be recovered so that the customer has peace of mind. Further, if corruption is in the I/O subsystem or the bus, it will be isolated.

Languages and Tools

Summarized by Asia Slowinska (asia@few.vu.nl)

Design and Implementation of an Embedded Python Run-Time System

Thomas W. Barr, Rebecca Smith, and Scott Rixner, Rice University

Even though there are dozens of microcontrollers around us—e.g., in cars, appliances, and computer electronics—the programming environments and runtime systems for them are extremely primitive. As a result, programming these devices is difficult. To address these issues, Thomas Barr presented Owl, a project which aims to let developers “build a toaster in Python.” Owl is a Python development toolchain and runtime system for microcontrollers. It also includes an interactive development environment, so that a user can connect to a device, and type Python statements to be executed immediately. As a result, experimenting with and programming microcontrollers becomes a much simpler task.

Microcontrollers come with limited resources: e.g., 64–128 KB of SRAM, and up to 512 KB of on-chip flash. These constraints require that Python code be executed with low memory and speed overheads. During his presentation, Barr discussed two of the features of Owl that make this possible. First, he explained how a compiled Python memory image is executed directly from flash, without copying anything to SRAM. One of the challenges here is to represent compound objects in such a way that they do not contain references to other objects—only then can they be used directly without an extra dynamic loading step. The next feature concerned native C functions that are called from Python to, for example, access peripherals. Owl provides a mechanism that wraps the C functions automatically, so that a programmer does not need to bother with converting Python objects into C variables, and vice versa. A full description of the Owl architecture is in the paper, and the authors can be reached at embeddedowl@gmail.com.

To demonstrate that the Owl system is practical, Barr showed a video of an autonomous RC car that uses a controller written entirely in Python. The car successfully detected and avoided obstacles as it zoomed around a room. A full description of the architecture of Owl can be found in the paper, and the authors can be reached at embeddedowl@gmail.com.

A questioner wondered how Owl provides access to some sort of global notion of time. Barr said that the virtual machine provides a function call that returns the number of milliseconds since the virtual machine booted. Rik Farrow asked how Owl makes interacting with peripherals simpler for a programmer. Barr explained that the embedded Python interpreter allows the programmer to interactively probe the device. Thus it becomes easy to tell whether a piece of code works as expected.

AddressSanitizer: A Fast Address Sanity Checker

Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitriy Vyukov, Google

Even though memory corruption bugs have been known about and fought for years, no comprehensive tool to detect them is available. To address this problem, Konstantin Serebryany presented AddressSanitizer, a memory error detector for C/C++ programs, which prevents out-of-bounds memory accesses and use-after-free bugs. The authors invite others to try it out. It is publicly available at http://code.google.com/p/address-sanitizer/.

AddressSanitizer is a compiler-level solution—it instruments the protected program to ensure that memory access instructions never read or write, so called, “poisoned” red zones. Red zones are small regions of memory (currently 128 bytes) inserted in-between any two stack, heap, or global objects. Since they should never be addressed by the program, an access to them indicates an illegal behavior. This policy prevents sequential buffer over- and underflows and some of the more sophisticated pointer corruption bugs. To deal with heap-use-after-free errors, AddressSanitizer marks a freed memory region as “poisoned.” Until this region is allocated again, any access to it causes an alert. AddressSanitizer uses its own tailored instrumentation of malloc and free, which keeps a released memory region in “quarantine” for as long as possible. By prolonging the period in which the memory buffer is not allocated again, it increases the chances of detecting heap-use-after-free bugs.

AddressSanitizer scales to real-world programs, and the developers at Google have been using it for over a year now. It has detected over 300 previously unknown bugs in the Chromium browser and in third-party libraries, 210 of which are heap-use-after-free bugs. The tool has a fair amount of overhead—it incurs 73% runtime overhead for the SPEC CPU2006 benchmark, and almost none for the I/O intensive Chromium browser.

During his presentation, Serebryany challenged the audience and hardware companies to attempt an implementation of AddressSanitizer in hardware. Rik Farrow asked what instruction would have to be added. Serebryany explained that a hardware version of the check which is performed on memory accesses—to ensure that the accessed memory is not poisoned—would be welcome. It would both improve performance and reduce the binary size. Since the current implementation of AddressSanitizer builds on the LLVM compiler infrastructure, the next questioner asked if Google plans to port it to gcc. Serebryany replied that they have already a version which can successfully compile the SPEC CPU2006 benchmark, but it is not fully fledged yet.

For the complete 2012 USENIX Annual Technical Conference report and summaries from HotCloud ’12, HotPar ’12, HotStorage ’12, and the panel at our first Women in Advanced Computing Summit, visit: www.usenix.org/publications/login.