usenix ;login:

FEBRUARY 2013 VOL. 38, NO. 1

ELECTRONIC SUPPLEMENT

Conference Reports

10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)

Hollywood, CA
October 8–10, 2012

Opening Remarks

Summarized by Rik Farrow (rik@usenix.org)

Program Co-chair Amin Vahdat opened the conference, telling the audience that this year's attendance was high, close to but not greater than the record for OSDI. Vahdat explained the review process: 25 out of 215 papers were accepted, producing 1079 written reviews. Authors were sent reviewers' comments so that they could address any concerns about the research before they submitted their final versions of their papers.

Co-chair Chandu Thekkath presented the two Jay Lepreau Best Paper awards to the 26 authors of "Spanner: Google's Globally-Distributed Database" for Best Paper, and to Mona Attariyan, Michael Chow, and Jason Flinn for "X-ray: Automating Root-Cause Diagnosis of Performance Anomalies in Production Software," the Best Student Paper.

The UCSC Cancer Genomics Hub

David Haussler, University of California, Santa Cruz (adjunct at UCSF & Stanford)

Summarized by David Terei (davidt@scs.stanford.edu)

Twelve years ago, two teams raced to sequence the human genome. Today, DNA sequencing technology is outpacing Moore's Law; it is the dawn of personal genomics. The first human genome cost more than $100 million to sequence, while in 2013, sequencing will cost $1,000 per personal genome.

Instead of simply looking at one reference genome to understand diseases, imagine looking at millions—this is where a lot of medical research will head. Cancer will be the main target as the disease is caused by changes to the human genome and is highly individual. There is also a willingness to try new ideas; the rest of the medical community is generally slower moving, said Haussler.

Thus, genomes are the key to the future of cancer treatment. A patient's genome will be compared to a database of other genomes to devise a treatment. Drugs targeted to the individual are needed and feasible.

The Cancer Genome Atlas is currently the largest such database, containing around 10,000 entries. Each entry consists of a biopsy of the tumor and normal tissue, with both sequenced and the differences analyzed and mapped to the first human genome reference to account for normal and abnormal variations between individuals. Generally speaking, there are 3 to 4 million normal differences between individuals; this variation among us is referred to as a persons "germ line." For sequenced genomes, a variety of mutations between the tumor and healthy tissue can be detected, including point mutations, deletions, duplications, inversions, and shifts.

The analysis of these mutations and the sequenced genome data itself are now being stored in the UCSC Cancer Genomics Hub (CGHub). CGHub provides a secure platform to host the confidential data and manage access to it for authorized researchers. The platform also supports an app-like model for authorized software that provides analysis and visualization tools to researchers. Each entry (patient) is around 100 GB when compressed; CGHub is currently designed for 50,000 genomes and holds 24,000 files from 5,500 cases. No collocated computing power is provided for now.

The analysis of these files is an interesting and challenging field for the systems and machine learning fields. Ideally, we would have large-scale, automated discovery of diagnostic signatures. (Think of how the prediction of treatment outcome would be improved if based on genetic data!) Here is where the systems community can help, by providing information on (1) how to handle big data, (2) how to analyze the data, (3) how to predict outcomes based on this analysis/data, and (4) how to make treatment recommendations from all of this.

Jeanna Matthews (Clarkson) asked if the benchmark challenge was available yet. Haussler replied that they will be announcing the benchmark challenge soon: read in the raw DNA and output how it was mutated. Ken Yocum (UCSD) asked Haussler to comment on consent from patients and privacy concerns for getting data into the database. Haussler said they were about to release some data without restrictions. In general, you need to apply for permission with NIH to gain access and abide by its privacy policy/expectations (prove you will do valuable research). Amin Vahdat (UCSD) asked about the cost for analyzing DNA data. Haussler replied that if the cost of sequencing is as low as $500 and the cost of analysis is $2000, there will be enormous pressure to also drive down the computational cost. They need better algorithms and novel techniques to accomplish this. For example, to compare every piece of DNA with 50 others currently takes a few weeks. Bryan Ford (Yale) asked whether Haussler could describe the major computational and storage roadblocks. Haussler said, in a word, I/O. Currently, individuals are writing small, isolated tools that create lots of intermediate files.

Big Data

Summarized by Jim Cadden jmcadden@bu.edu

Flat Datacenter Storage

Edmund B. Nightingale, Jeremy Elson, and Jinliang Fan, Microsoft Research; Owen Hofmann, University of Texas at Austin; Jon Howell and Yutaka Suzue, Microsoft Research

Jeremy Elson presented the Flat Datacenter Storage (FDS), a datacenter-scale blob store that has the agility and conceptual simplicity of a global store without the usual performance penalty. The novel approach taken by FDS to alleviate the network bottleneck is to multiplex the application's I/O across the available throughput and latency budget of the disks within the system.

Jeremy began the talk with a conceptual introduction to a "little data" platform—a highly utilized, tightly coupled multicore machine. This commonplace example illustrated the inherent problem with big data computation in that our traditional machine architectures do not scale. FDS attempts to provide the essential properties of little data platforms with the scale and performance necessary for big data application. This is realized through a novel combination of three attributes: a simple scalable blog store, decentralized metadata management, and a full bisection bandwidth CLOS network with novel distributed traffic scheduling. The performance gains of FDS come at the cost of the requirement of additional underlying hardware, in specifically the 1:1 matching between I/O and network bandwidth.

In the experimentation results, FDS showed read/writes to remote disks at up to 2 GBps—faster than most systems write locally. In addition, intra-disk high-speed communication of FDS allows for impressive data recovery benchmarks, with over 600 GB of data being recovered in 34 seconds within a 1000 node cluster. Applications built atop FDS are able to achieve world-record-breaking performance. MSR trumped Yahoo!'s 2009 record at Minute Sort benchmark by sorting data at a 15x efficiency improvement over the existing record.

Geoff Kuenning (Harvey Mudd) asked if the replication communication of the coherence protocol would present a potential bottleneck. Jeremy agreed that a replicated cluster would incur some necessary cost, but the applications flexibility to extend blobs and lazy disk space allocation will help alleviate this cost. Someone asked whether the authors had compared the performance of FDS against other commercial high-end storage systems (e.g., EMC, Hatachi, etc.) and how they expected FDS to scale further. Jeremy explained that they do not have the opportunity to do a 1:1 datacenter-scale comparison and, in addition, the linear scaling characteristics of FDS should allow for a scale of up to tens of thousands of nodes.

PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

Joseph E. Gonzalez, Yucheng Low, Haijie Gu, and Danny Bickson, Carnegie Mellon University; Carlos Guestrin, University of Washington

Joseph Gonzalez presented PowerGraph, a framework for graph-parallel computation on natural graphs. PowerGraph was shown to produce order-of-magnitude computation improvement on natural graphs over the existing graph-parallel abstraction frameworks such as Pregel and GraphLab version 1. PowerGraph is now integrated into GraphLab 2.1 and released under the Apache license.

Joseph began by introducing natural graphs, graphs that, by definition, are derived from real-world phenomena, like a Twitter connection graph. Natural graphs are commonplace in machine learning and data mining problems throughout science. A distinct trait of natural graphs is their highly skewed power-law degree distributions that create a star-like graph motif. An example illustrating this was that of President Obama's twitter account and his many followers.

PowerGraph changes the model for structuring the parallel computation on a graph by splitting up a high-degree vertex and distributing it across machines. PowerGraph's gather, apply, and scatter (GAS) technique further enables the work to be divided, computed in parallel, and sent to a "master" machine where changes are applied on the master and synced to mirror nodes. The GAS method was shown to be applicable to all existing graph processing systems by a new theorem that states any edge cut can be reconstructed as a vertex-cut. Preprocessing and greedy-cut techniques can further increase the performance of PowerGraph.

In the experimental results, PowerGraph was shown to provide an order-of-magnitude performance increase over previous frameworks in both throughput and runtime. The scalability of PowerGraph was illustrated through PageRank iterations on the billion-node Yahoo! Altavista Web Graph, spending only seconds running across on 64 Amazon EC2 HPC nodes.

Terence Kelly (HP Labs) noticed that the graphs used to gather experimental data were small enough to fit into main memory, and proposed that a comparison between PowerGraph and a simple straightforward serial process would be interesting. Joseph agreed that PowerGraph can be beaten by a single core to a point, but it makes great strides on larger graphs and in the cloud. Aapo Kyrola (CMU) inquired about the difference between the approach of PowerGraph and previous vertex partitioning techniques. Joseph explained that the objectives of PowerGraph include an asynchronous operation and the ability to transform edge data. Aapo also made the same point as Terence Kelly, that it is often possible to compute graph problems on a single host, without partitioning.

GraphChi: Large-Scale Graph Computation on Just a PC

Aapo Kyrola and Guy Blelloch, Carnegie Mellon University; Carlos Guestrin, University of Washington

Aapo Kyrola introduced GraphChi, a natural graphic processing framework designed to compute on a single desktop PC via appropriate use of data structure and algorithms. GraphChi was written in 8000 lines of C code and is also available as a Java implementation.

Aapo began by setting the assumption that most large-scale natural graphs (e.g., Facebook social connections) have billions of edges yet can be reasonably stored on a single hard drive and, therefore, may not need the added overhead and cost required by the cloud if the computation can be handled on a single machine. The GraphChi model, similar to that of PowerGraph, is designed to exploit the inherent characteristics of natural graphs. Perhaps more of a fan of music than politics, Aapo used Lady Gaga's Twitter followers to illustrate the high-degree vertices trait of natural graphs.

One way in which GraphChi enables sequential scalable graph computation on a single machine is through increased I/O performance gained through optimizations made to alleviate random-access reads on disk. The Parallel Sliding Window (PSW) works by loading subgraphs into memory, one at a time, running computation, and returning that subgraph to disk. Heavy preprocessing is required on the graph to sort the vertex and edges in a way that both in and out edges of a directed subgraph can be extracted per load interval.

In the experimental results, Aapo showed that GraphChi performs reasonably well with large scale Big Data graph computations on a single Mac Mini machine. For problems dealing with complex computational issues, a 4x speedup was recorded by running GraphChi across four cores. The same speedup was observed with problems involving high I/O as GraphChi would saturate the I/O lines with only two concurrent threads. Graphs with billions of edges required less than an hour of preprocessing.

Mohit Saxena (Wisconsin) asked if their research compared PWS with OS techniques for memory mapping or SSD as a cache. Aapo explained that since the graph is passed over once in its entirety (and then flushed), OS caching techniques don't really apply.

Privacy

Summarized by Edmund Wong (elwong@cs.utexas.edu)

Hails: Protecting Data Privacy in Untrusted Web Applications

Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, David Mazières, and John C. Mitchell, Stanford University; Alejandro Russo, Chalmers University

Web platforms, such as Facebook, currently host many third-party apps that access private user data. It is hard to place trust in third-party app code; developers of such apps, due to malice or ignorance, may leak private data. However, even if developers are well-meaning, it is hard to trust their app, as building secure Web apps is difficult. Typically, developers implement security policy in an error-prone fashion, by injecting if-statements in application logic. A typical platform protects user data by allowing users to decide whether an app gets access to data, but the platform does not control how the app uses said data.

To address this problem, Deian Stefan from Stanford University presented Hails, a Web platform framework that enables security policies to be explicitly specified alongside data. Hails aims to be deployable today, usable by non-security developers, and suitable for building extensible Web platforms. In Hails, the trusted platform provider hosts untrusted apps and enforces security through language-level information-flow control (IFC) techniques. Stefan argued that, unlike Hails, previous IFC systems, such as Aeolus, HiStar, Nexus, or Jif, provide no guide for structuring apps, require policies that are hard to write, are not appropriate for dynamic systems such as the Web, and/or require modifications to the entire app stack.

Hails introduces a new design pattern, Model-Policy-View-Controller (MPVC), which is an extension of Model-View-Controller. In MPVC, the model-policy consists of the data model and the policy associated with the data. The policy follows the data as it flows through the system and specifies where data can flow. The view-controller components provide application logic and user-interface elements; they do not implement security policy and are not trusted by users. View-controller components invoke model-policy components to store/fetch user data, and a Hails runtime enforces that the policy is followed end-to-end.

Hails is implemented as a Haskell library that enables quick turnaround on API design and allows developers to use their existing tools and libraries. Part of the library is the Hails runtime which provides an HTTP server that runs the view-controller. To demonstrate Hails in action, Stefan presented GitStar, a Web site for hosting source code much like GitHub, except the platform is provided by Hails and untrusted apps run atop this platform, unlike the monolithic structure of GitHub. In GitStar, the model-policy consists of data on projects and users, and third-party developers can build apps (view-controllers) to implement functionality, such as a code viewer or a wiki. Hails ensures that these apps cannot leak data even if the apps access project and user data. Stefan evaluated the usability of Hails by asking five developers to implement apps using Hails. These developers thought that Hails greatly simplified the process of writing security policies and securing their apps. Stefan also showed that

Hails was faster than Sinatra, another Ruby Web application framework, for small Ruby apps but slower than Apache running PHP.

The first questioner asked how Stefan evaluated the usability of Hails for users. Stefan reiterated that the five developers (including one high-school student) who were asked to develop on Hails were successful in doing so; these developers found that Hails greatly simplified the process. Stefan was particularly excited by this result because he felt that developers using other IFC systems were typically experts in IPC, not Web developers. Jonas Wagner (EPFL) asked whether there were obstacles to applying the techniques used in Hails to a framework in Ruby on Rails or Python and whether policy can be enforced without modifying the language runtime. Stefan replied that other languages were considered, but Haskell was chosen due to control over side effects. Stefan said that for any other language, the compiler must be modified to support Hails. However, Hails currently allows developers to write apps that call untrusted (Linux) executables and referred to the paper for a further discussion on the use of untrusted executables within Hails. Peter Goodman (U of Toronto) asked which compiler was used with Hails; Stefan replied that GHC was used.

Eternal Sunshine of the Spotless Machine: Protecting Privacy with Ephemeral Channels

Alan M. Dunn, Michael Z. Lee, Suman Jana, Sangman Kim, Mark Silberstein, Yuanzhong Xu, Vitaly Shmatikov, and Emmett Witchel, The University of Texas at Austin

Alan Dunn presented Lacuna, whose goal is to provide forensic deniability: no evidence is left for a non-concurrent attacker once the program has terminated. Dunn argued that current approaches, such as private browsing and secure deallocation, still leave traces of private data because these approaches are hindered by the lack of proper system support. Lacuna's goals are to protect a user's privacy even under extreme circumstances—even if the machine is compromised at the root level or is physically seized—while maintaining usability. Lacuna supports running private applications (i.e., those that preserve a user's privacy under Lacuna) alongside non-private applications; supports a wide variety of private applications; and has reasonable overhead that is only incurred on private applications.

Lacuna achieves these goals by running applications inside erasable program containers and providing privacy-preserving I/O channels from these containers to hardware. The erasable program containers are virtual machines (VMs), where I/O can be intercepted by the virtual machine monitor (VMM) as necessary. Lacuna provides several I/O channels: disk I/O is encrypted before leaving the VMM and decrypted within the VMM upon being read back. For hardware that supports hardware virtualization (e.g., USB, network), Lacuna allows the driver running within the erasable program container to control and communicate directly with the hardware, bypassing the host OS in the process. This approach requires no modifications to host drivers, and any code running outside the erasable program container never sees unencrypted data. For hardware that does not support hardware virtualization (e.g., the graphics card), Lacuna provides software proxies that are placed close to where data is pushed or pulled from hardware, in host drivers or even on a graphics card. When a contained application performs an I/O operation with this type of hardware, the VMM passes the data for the operation to/from that hardware's associated software proxy via a cryptographically secured channel. This approach requires no modification to guest applications, and any residual data that may remain in the host OS is cryptographically erased by deleting the encryption/decryption keys when the channel is no longer in use.

Dunn then described how he evaluated Lacuna, which was implemented as a modified version of QEMU-KVM (a virtual machine monitor) running atop a modified version of Linux as the host OS, to show that Lacuna met its privacy and usability goals. In one experiment, Dunn injected random tokens into peripheral I/O paths and scanned memory to see whether these tokens could be located in various applications and in the OS in order to gauge what code holds on to sensitive data. Without Lacuna, these tokens are almost always found; with Lacuna, the tokens are never found. While the latency incurred when switching between private and non-private applications is low, Lacuna incurs higher overhead for performing USB I/O operations due to interactions between the guest and host OSes; Lacuna reduces this overhead to some degree by eliminating extra disconnections that occur when the guest OS performs USB initialization. Finally, Dunn showed that while Lacuna incurs higher CPU utilization, applications experience minimal slowdown, partly thanks to the availability of hardware AES support.

Fitz Nolan wondered whether external parties could potentially force the user to decrypt sensitive data and whether usage of Lacuna would imply guilt. Dunn said that Lacuna prevents users from incriminating themselves since all traces of private applications are removed (at least cryptographically), and it would be up to the courts whether users could get in trouble for not being able to decrypt their data. Dunn also disagreed that users would only use Lacuna if they had something to hide. Someone asked how unencrypted data in the device buffers were handled. Dunn replied that Lacuna uses public APIs to clear out as much data as possible, but it is possible that device buffers keep some data around. Peter Desnoyers (Northeastern) asked whether the graphics benchmark unfavorably favors Lacuna because Lacuna's implementation could potentially avoid some steps that would otherwise have to be taken. Dunn said they did not exploit this shortcut in their evaluation. Finally, someone from Microsoft asked how difficult it was to modify device drivers for Lacuna and about the complexity of supporting 3D APIs. Dunn responded the difficulty varies per subsystem and that often one can capture a lot of devices with a single modification; at the moment, Lacuna does not support 3D acceleration.

CleanOS: Limiting Mobile Data Exposure with Idle Eviction

Yang Tang, Phillip Ames, Sravan Bhamidipati, Ashish Bijlani, Roxana Geambasu, and Nikhil Sarda, Columbia University

Yang Tang began by saying that CleanOS provides new OS abstractions for protecting sensitive data on mobile devices. The mobile nature of these devices results in their being easily stolen, seized, or lost. Because the OSes running on these devices are not designed to protect sensitive data, mobile devices accumulate large amounts of sensitive information that can be accessed by anyone who has physical access to the device. Tang showed that 13 out of the 14 Gingerbread apps his research group studied kept sensitive data in cleartext either in memory or persistent storage. Moreover, Tang cited that many users do not lock their devices or use poor passwords.

Tang proposed CleanOS, a mobile Android-based OS that rigorously protects sensitive data in anticipation of device theft/loss and allows the user to know exactly what data is exposed. CleanOS leverages three critical insights: (1) sensitive data is often rarely used (e.g., an email password is only needed when sending or fetching new messages); (2) mobile apps often already contain cloud components that store data; and (3) mobile devices are almost always connected via WiFi and cellular connections. In CleanOS, sensitive data is encrypted and stored in a sensitive data object, or SDO. When access to a SDO is needed, CleanOS contacts a trusted cloud component to retrieve the decryption key needed to access the SDO. CleanOS uses taint-tracking on the local device to track accesses to sensitive data located in RAM and stable storage. When an SDO has not been accessed in a while, CleanOS will automatically cryptographically evict the SDO by securely deleting the decryption key off the local device. By minimizing the amount of sensitive data on a user's local device and shifting the protection of sensitive data to the cloud, CleanOS offers the ability to audit or limit the amount of exposure or access to said data; access to sensitive data can be completely revoked if the user's device is stolen.

CleanOS is implemented as a modified version of Android/TaintDroid that uses a CleanOS cloud service on Google App Engine. Tang described how an email app that his research group implemented within CleanOS reduced exposure of sensitive data by roughly 90% without modification (CleanOS automatically puts SSL-related state, passwords, and user input into SDOs). Modifying the app to use SDOs reduced content exposure to 0.3% that of the unmodified app. Moreover, Tang showed that auditing is very precise when the app is modified to support SDOs and can still be precise with specific types of data (e.g., passwords) even when the app is not. Finally, Tang showed that the overheads of CleanOS are largely unnoticeable over WiFi. On 3G, the overheads are more significant but can be made reasonable through a series of optimizations that Tang proposed, including batching evictions and retrievals of decryption keys.

Mark Silverstein (UT Austin) asked what the power consumption overhead of using CleanOS was. Tang responded by stating that while CleanOS adds some overhead (less than 9% overall), this overhead is largely dwarfed by the power consumed by the screen. Jason Flinn (Michigan) asked about the fundamental tradeoff between the performance benefits associated with caching and the security and granularity of caching. Tang replied that this is a policy decision that the user can configure. Stefan Bucur (EPFL) asked whether eviction continues when the device is taken offline. Tang responded that in the case of short-term disconnections, CleanOS can delay the eviction of SDOs by a bounded amount of time; for long-term disconnections, CleanOS can be configured to hoard keys before being disconnected. Finally, Bryan Ford (Yale) asked whether taint explosion is a problem. Tang said that in his experience it was not, and that running their email app for 24 hours resulted in only about 1.8% objects being tainted.

Mobility

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

COMET: Code Offload by Migrating Execution Transparently

Mark S. Gordon, D. Anoushe Jamshidi, Scott Mahlke, and Z. Morley Mao, University of Michigan; Xu Chen, AT&T Labs—Research

Mark Gordon began with a discussion of offloading. He observed that mobile device resources are limited in terms of computation, energy, and memory; yet mobile devices are often well connected to the network. COMET explores the question of how to add network resources to mobile device computations transparently. Previous projects have explored the capture-and-migrate paradigm, such as CloneCloud and MAUI. COMET distinguishes itself from these approaches by enhancing support for multithreaded environments and synchronization primitives. Mark also noted that COMET's fine granularity of offloading efficiently handles functions with work loops, since resources can be offloaded in the middle of a method. At a high level, COMET merges the motivation of offloading—to bridge the computation disparity among nodes in a network—with the mechanism of distributed shared memory, which provides a logically shared address space.

Mark explained that distributed shared memory (DSM) is traditionally applied in cluster environments, which have low latency and high throughput. However, COMET relies on wireless communication between two endpoints. COMET implements a simple field-based DSM scheme, in which dirty fields are tracked locally, and only dirty fields are transmitted. COMET DSM leverages the Java memory model, which is field based and which specifies a happens-before partial ordering among all memory accesses.

VM synchronization is used to establish the happens-before relationship between two endpoints. VM synchronization is a directed operation between a pusher and a puller, and is responsible for synchronizing thebytecode sources, Java thread stacks, and Java heap. Thus, thread migration is implemented as a push VM synchronization operation. Mark noted that VM synchronization is designed to be recovery safe; the client is always left with enough state to resume operation in the event that the server thread is lost.

Mark was asked about the user input component of most Android apps. He replied that certain application types will not benefit from offloading, including applications with a lot of user interactions. However, there are applications, such as turn-based games, and applications with some type of kernel computation, that would benefit from using COMET. COMET may also open up new types of applications. Mark was also asked to restate the differences between COMET and CloneCloud. Mark noted that COMET provides complete support for offloading multiple threads—threads never have to block to wait for remote state. He also noted that COMET can offload within a method. Arjun Roy (UCSD) asked about I/O requests, and Mark responded that I/O requests translate to native functions. They did not have time in this paper to try to virtualize the FS like CloneCloud.

AppInsight: Mobile App Performance Monitoring in the Wild

Lenin Ravindranath, Jitendra Padhye, Sharad Agarwal, Ratul Mahajan, Ian Obermiller, and Shahin Shayandeh, Microsoft Research

Lenin Ravindranath exclaimed that developers are interested in two things: where user-perceived delays crop up, and, when they do, what is the bottleneck? To answer these questions, developers must manually instrument apps, which poses a significant barrier for the average developer. AppInsight significantly reduces the barrier for monitoring performance in the hands of users. It is completely automatic, requiring no effort from developers; AppInsight does not require source code, runtime, or OS modifications. A developer simply writes an app, AppInsight performs binary instrumentation, and the developer submits the instrumented app to the app store.

Lenin explained that a fundamental problem for automatic app instrumentation is that modern apps are highly interactive, UI-centric programs, which are written using very asynchronous programming patterns. With synchronous code, the user-perceived delay can be calculated by observing the beginning and end of functions. With asynchronous code, background threads process individual tasks. The user-perceived delay includes the time for the entire execution, and to measure this, the monitor must track time across thread boundaries. However, AppInsight does not modify the runtime, so it has no context for executing threads. It must have a way to know which asynchronous call is responsible for invoking each thread. Lenin defined a user transaction as beginning with a UI manipulation, and ending with the completion of all synchronous and asynchronous threads triggered by that manipulation. The critical path is the bottleneck path through a user transaction, where speeding up the path will reduce the user-perceived delay. AppInsight automatically instruments apps to track user transactions and the critical path.

In additional to performing critical path analysis for each transaction, AppInsight provides aggregate analysis. Aggregate analysis can give developers additional insight into what factors cause delay. AppInsight can group transactions and use statistical analysis to identify the root causes of variability, identify group outliers, and highlight common critical paths. AppInsight can also be used to understand app failures in the wild—since entire transaction graphs are tracked, developers can walk backwards through the path to figure out which user event triggered the exception. All the analysis is made available to developers through a Web-based tool.

Frank Lih asked if AppInsight can be used to identify performance problems caused by other applications, perhaps due to competition for resources. Mark replied that AppInsight can be imagined as the first step in finding a performance problem. He was next asked about their evaluation, and how experiences might change when applications with thousands of users are monitored. Lenin responded that more interesting problems will be identified as more diverse device configurations and environmental conditions are encountered.

OSDI 2012 Poster Session 1

Summarized by Peter Gilbert (petergilbert@gmail.com)

Be Conservative: Enhancing Failure Diagnosis with Proactive Logging

Ding Yuan, University of Illinois at Urbana-Champaign and University of California, San Diego; Soyeon Park, Peng Huang, Yang Liu, Michael M. Lee, Xiaoming Tang, Yuanyuan Zhou, Stefan Savage, University of California, San Diego

Diagnosing production failures is difficult because the execution environment and user inputs often cannot be reproduced exactly. Log messages are often the best available resource, reducing diagnosis times by 1.4x–3.0x for failures in software such as Apache, PostgreSQL, and Squid. However, the authors found that log messages were printed in only 43% of failures in a survey of real-world errors. Further examination revealed that 77% of failures had an explicit and generic error condition. They present ErrLog, an automated tool that analyzes and instruments source code to insert logging for error conditions. They are able to reduce the diagnosis time by 60% for real-world failures while adding a modest 1.8% runtime overhead.

ϖBox: A Platform for Privacy-Preserving Apps

Sangmin Lee, Edmund L. Wong, Deepak Goel, Mike Dahlin, and Vitaly Shmatikov, The University of Texas at Austin

There is growing concern about smartphone apps mishandling users' privacy-sensitive data. Instead of relying on untrustworthy apps to properly handle personal data, the authors propose shifting the responsibility to a platform that isolates apps from user data. ϖBox provides a sandbox that spans the device and the cloud and controls storage and communication of sensitive data. Privacy policies are configured based on an app's functionality: for example, an app that uses location information for localization is prevented from releasing that data, while usage statistics and advertising data are allowed to be released only through an aggregate channel that respects differential privacy.

Diagnosis-Friendly Cloud Management Stack

Xiaoen Ju and Kang G. Shin, University of Michigan; Livio Soares, Kyung Dong Ryu, and Dilma Da Silva, IBM T.J. Watson Research Center

The authors argue that it is important for a cloud management layer to be both easy to use and reliable, easy to diagnose and debug. To address this need, they propose logging message flows and building diagnostic tools to analyze this information. Examples of proposed tools include a tool for detecting anomalous message flows, a testing tool that can inject faults to explore recovery logic, and a replay tool to run tests offline.

Processing Widely-Distributed Data with JetStream

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

This work aims to enable queries over data sets that are distributed over wide areas, such as smart grid monitoring data or traffic statistics from Web services. Goals include moving computation to data when possible, adapting to network variation, using approximations when bandwidth limitations prohibit exact answers, and allowing users to configure algorithms. To support these features, they present JetStream, which combines data cubes from online analytical processing (OLAP) with techniques from streaming databases. Advantages of their approach include efficient per-cube durability, easy specification of approximations through dimension hierarchies, explicit data movement via streams, and compatibility with open-ended data like logs.

C3A: Client/Server Co-Verification of Cloud Applications

Stefan Bucur, Johannes Kinder, George Candea, EPFL

Cloud applications are increasingly (1) split among multiple administrative domains and (2) heterogeneous, consisting of components built using different programming languages. These characteristics make cloud applications difficult to test and verify. To address this problem, the authors propose a technique called federated symbolic execution, in which specialized symbolic execution engines for different languages share a common symbolic data representation. Symbolic execution is driven by request specifications provided by the developer using an API in the target language. Testing runs as a cloud service, and properties defined in request specifications are verified along execution paths.

Hails: Protecting Data Privacy in Untrusted Web Applications

Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, David Mazières, and John C. Mitchell, Stanford University; Alejandro Russo, Chalmers University

This work addresses the problem of protecting users' private data spread across inter-connected Web applications. The authors argue that existing APIs force users to resort to coarse-grained access control and to choose between privacy and features. For example, a user must decide between granting an application access to her Facebook data, at which point she forfeits control of the data, or not using the application at all. The authors present the Hails multi-application Web platform as an alternative. Hails executes each Haskell application inside a jail and controls whether private data can be released by the application. Language-level information flow control is used to track private data as an application executes. Policies specifying how private data can be shared are stored alongside the data itself and enforced globally across all applications.

ContextJob: Runtime System for Elastic Cloud Applications

Wei-Chiu Chuang, Bo Sang, Sunghwan Yoo, Charles Killian, and Milind Kulkarni, Purdue University

A key advantage offered by the cloud for applications is elasticity, or the ability to scale dynamically with demand to take advantage of additional resources. However, the authors argue that it is difficult for programmers to reason about application semantics and ensure correctness when designing elastic applications. To alleviate this problem, they propose a programming model in which developers write seemingly inelastic code and elasticity is handled by the runtime system. The programmer works with the simple abstraction of an event queue.

What Does Distributed Computing Look Like on a Multicore Machine?

Stefan Kaestle and Timothy Roscoe, ETH Zürich

This work explores how to achieve high performance for parallel applications running on complex multicore machines. This can be difficult due to the challenges of ensuring efficient access to global state such as database tables. Both hardware characteristics and software requirements must be considered. The authors advocate an approach that (1) abstracts global state to support agile placement and access, and (2) chooses among distributed algorithms dynamically. In ongoing work, they plan to explore how to best abstract global state and to quantify the cost of automating these choices compared to hand-tuned implementations.

X-ray: Automating Root-Cause Diagnosis of Performance Anomalies in Production Software

Mona Attariyan, University of Michigan and Google, Inc.; Michael Chow and Jason Flinn, University of Michigan

This work focuses on troubleshooting performance problems in complex production software. While profiling and logging can reveal which events occurred, determining how and why the events affected performance is often a challenging manual task. The authors present X-ray, a tool for automating this process by attributing performance to specific root causes. X-ray does so by assigning costs to operations such as system calls and applying taint tracking to connect these operations to a root cause. The time-consuming analysis is completed offline using deterministic replay, adding an online overhead of only 1–5%. In an evaluation using real performance issues in software such as Apache and PostgreSQL, X-ray correctly ranked the actual root cause first or tied for first in 16 out of 17 cases.

Toward Emulating Large-Scale Software Defined Networks (SDN)

Arjun Roy, Danny Yuxing Huang, Kenneth Yocum, and Alex Snoeren, University of California, San Diego

To facilitate developing emerging software defined network (SDN) technologies, testing platforms are needed for experimenting with large-scale SDNs. The authors propose an architecture consisting of multiple ModelNet emulator instances to increase bandwidth. Challenges include how to maximize bandwidth for each emulator host and how to account for the effects of different OpenFlow implementations from different vendors. They propose profiling real OpenFlow switches to quantify idiosyncrasies and then replicating them in emulated switches.

Rearchitecting System Software for the Cloud

Muli Ben-Yehuda and Dan Tsafrir, Technion—Israel Institute of Technology

The authors observe that traditional operating systems are poorly suited for cloud environments where users pay per-use for a number of reasons: applications are constrained by kernel abstractions and implementation choices that are hidden by design, and applications share a single I/O stack and device drivers. The authors present an alternative platform called nom that takes advantage of architectural support for virtualization to provide each application direct and secure access to its own I/O device. This enables the use of I/O stacks and device drivers optimized for specific applications. Applications can also change behavior to adapt to changing resource availability and pricing.

Who is Going to Program This?

Marcus Völp, Michael Roitzsch, and Hermann Härtig, Technische Universität Dresden

This work anticipates challenges programmers will face when developing for future heterogeneous manycore machines. Potential components include powerful cores "fused" from multiple smaller cores, redundant cores to handle specific hardware errors, and specialized accelerator cores. The authors argue that there is a mismatch between new properties (two-way mediation between hardware and applications, adaptive software parallelism, reconfigurable hardware, and spatial data placement) and current application characteristics (hardcoded threads, ad hoc use of accelerators, and opaque data use). They propose an "Elastic Manycore Architecture" that uses lambdas not threads for parallelism, queues for asynchronous work, runtime profiling, decentralized scheduling decisions, and an "execution stretch" metric to quantify utilization and efficiency. Cross-layer information is shared at interfaces between the OS, runtime, and applications.

Performance Isolation and Fairness for Multi-Tenant Cloud Storage

David Shue and Michael J. Freedman, Princeton University; Anees Shaikh, IBM T.J. Watson Research Center

The authors argue that existing cloud storage services, while implementing pay-per-use service on shared infrastructure, either offer no fairness or isolation, or assume uniform demand and are non work-conserving. They present a system for predictable shared cloud storage called Pisces. Pisces provides per-tenant weighted fair shares of system resources without sacrificing high utilization. The approach comprises four mechanisms: placement of partitions by fairness con-straints, allocation of local weights by tenant demand, selection of replicas using local weights, and weighted fair queuing. Evaluation results show that Pisces achieves nearly ideal fair sharing, performance isolation, and robustness to changes in demand, while imposing an overhead of less than 3%.

Devirtualization: I/O Virtualization Based on Device Files

Ardalan Amiri Sani, Rice University; Sreekumar Nair, Nokia Research Center; Kevin A. Boos and Lin Zhong, Rice University; Quinn Jacobson, Nokia Research Center

Devirtualization is an approach for allowing the OS running in a guest virtual machine (VM) to directly use a device driver running in the host OS through a simple virtual device driver. The approach leverages the widely used device file interface, which provides a narrow and stable boundary for device drivers and already supports multiplexing among processes. Devirtualization can support many GPUs and input devices in guest VMs with minimal device-specific changes while imposing no user-perceptible latency. A virtual device file in a guest VM corresponds to the actual device file in the host, and file operations performed by the guest are forwarded to the host and performed by the actual device driver. Because guest file operations use separate guest virtual addresses, a hybrid address space is provided to bridge guest user space memory and host kernel memory.

Dune: Safe User-Level Access to Privileged CPU Features

Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazières, and Christos Kozyrakis, Stanford University

The authors argue that many applications could benefit from a safe, efficient way to directly access privileged CPU features such as page tables, tagged TLBs, and ring protection. Standard mechanisms like ptrace and mprotect are too slow and clumsy, while modifying the kernel for every application is risky and does not scale. Simply running each application in its own virtual machine (VM) is dismissed due to poor integration with the host OS and unacceptable overhead. Instead, Dune leverages architectural support for hardware-assisted virtualization but exports a process abstraction rather than a VM abstraction. A minimal kernel module manages virtualization hardware and interactions with the kernel, while the libDune library helps applications manage privileged CPU features. The authors demonstrate Dune's value by implementing a privilege separation facility, a sandbox for untrusted code, and a garbage collector.

Mercurial Caches: OS Support for Energy-Proportional DRAM

Asim Kadav, Rathijit Sen, and Michael M. Swift, University of Wisconsin—Madison

While DRAM is a significant contributor to power consumption, techniques that take advantage of unused DRAM to save power are limited. The authors propose mercurial caches to provide OS abstractions for low-power DRAM. Mercurial caches occupy portions of DRAM to put them in a low-power state. Challenges include how to dynamically resize mercurial caches without affecting application performance, how to ensure that mercurial caches do not appear as missing memory, and how to utilize mercurial caches when they are not completely turned off. Energy-aware page migration is needed for mercurial caches to be effective despite fragmentation of the physical address space. A preliminary evaluation using an analytical model shows that mercurial caches can achieve energy usage proportional to DRAM usage.

Herding the Masses—Improving Data/Task Locality in Hadoop

Bingyi Cao and Daniel Abadi, Yale University

The authors demonstrate that the default scheduler for map tasks in Hadoop, which greedily selects tasks with data nearby in FIFO order, can result in poor locality. They propose an alternative two-level sort algorithm consisting of a coarse-grained sort followed by a fine-grained sort, using only information about the local node and tasks. High efficiency is possible because only a few tasks need be sorted for each node.

Optimizing Shared Resource Contention in HPC Clusters

Sergey Blagodurov and Alexandra Fedorova, Simon Fraser University

This work focuses on performance problems in HPC clusters due to contention for shared multicore resources such as caches and memory controllers. The authors argue that HPC clusters must be made contention-aware to remedy this problem. They present Clavis-HPC, a contention-aware virtualized HPC framework. Tasks are classified as devils if they are memory-intensive with a high last-level cache miss rate, or turtles otherwise. The scheduling algorithm minimizes the number of devils on each node, maximizes the number of communicating processes on each node, and minimizes the number of powered-up nodes in the cluster. The schedule is enforced using low-overhead live migration.

OSDI Poster Session 1

Summarized by Lisa Glendenning (lglenden@cs.washington.edu)

Towards a Data Analysis Recommendation System

Sara Alspaugh, University of California, Berkeley; Archana Ganapathi, Splunk, Inc.; Randy Katz, University of California, Berkeley

This project proposes building a tool for semi-automated, user-guided exploration of arbitrary data sets. The proposed approach is to build a tool on top of Splunk, a platform for indexing and searching semi-structured time series data. Existing packages built on top of Splunk provide front-end analyses such as reports and dashboards, but each package is tailored to a specific class of data. The proposed tool would create and recommend front-end analyses for arbitrary data sets and iterate based on user feedback.

Nested Virtual Machines and Proxies for Easily Implementable Rollback of Secure Communication

Kuniyasu Suzaki, Kengo Iijima, Akira Tanaka, and Yutaka Oiwa, AIST: National Institute of Advanced Industrial Science and Technology; Etsuya Shibayama, The University of Tokyo

The objective of this work is to use fuzz testing to verify implementations of protocols for secure communication; the current target is TLS/SSL. This project's approach for verification depends on a rollback capability that the authors have implemented using nested virtual machines, proxies, and a control protocol. Work in progress includes developing a protocol fuzzing generator as a client component.

MiniStack: Operating System Support for Fast User-Space Network Protocols

Michio Honda and Felipe Huici, NEC Europe Ltd.; Luigi Rizzo, Universita di Pisa

The motivation of this work is to move the network stack into user space while approaching the high performance, isolation, and security of kernel- and hardware-based networking implementations. MiniStack builds on the high performance of netmap while addressing some of netmap's limitations: applications can snoop and overwrite each other's packets; applications can spoof the packet source address; and there is no mechanism to demultiplex received packets to the appropriate application. MiniStack extends VALE, a virtual Ethernet switch implemented as a Linux and BSD kernel module.

POD: Performance-Oriented I/O Deduplication for Primary Storage Systems

Bo Mao and Hong Jiang, University of Nebraska—Lincoln; Suzhen Wu, Xiamen University

Deduplication reduces I/O redundancy in primary storage systems but can lead to data fragmentation and resource contention. POD mitigates these problems by selectively deduplicating write requests and by dynamically adjusting the memory space between the index cache and the read cache based on the I/O accesses.

A Verified Kernel and Commodity Hardware

Yanyan Shen and Kevin Elphinstone, University of New South Wales, NICTA

Formally verified microkernels such as seL4 provide a strong foundation on which to build trustworthy systems, but commodity hardware is susceptible to transient faults such as silent memory corruption and bus errors. This project proposes using techniques such as redundant execution on multiple cores to detect and recover from hardware faults in the trusted software layer.

Closing the Gap Between Driver Synthesis and Verification

Alexander Legg and Leonid Ryzhyk, NICTA; Adam Walker, NICTA and University of New South Wales

Driver synthesis automatically generates a device driver implementation from a device and OS specification. This work proposes using a code template for the OS specification rather than a state machine, which is susceptible to state explosion and is hard to maintain. A code template requires some manually written code, but these code snippets undergo verification during the synthesis process.

Collaborative Verification with Privacy Guarantees

Mingchen Zhao, University of Pennsylvania; Wenchao Zhou, Georgetown University; Alexander Gurney and Andreas Haeberlen, University of Pennsylvania; Micah Sherr, Georgetown University; Boon Thau Loo, University of Pennsylvania

In distributed systems, nodes fail in a number of ways including misbehavior—e.g., violating protocol specifications. The goal of this work is to verify whether a node is behaving as expected without revealing sensitive information about non-faulty nodes. The approach is to generalize the collaborative verification techniques that were used in the SPIDeR system to verify the interdomain routing decisions of BGP systems. Ongoing work includes automatic generation of verification protocols via a new programming language.

The Ethos Project: Security Through Simplification

W. Michael Petullo and Jon A. Solworth, University of Illinois at Chicago

Existing operating systems have very high complexity, and this complexity limits the level of assurance possible. Ethos is an experimental, clean-slate OS with security as a primary goal. Ethos is designed to support the development and deployment of secure systems for both application developers and system administrators; for instance, Ethos provides a small set of carefully chosen system calls with high-level semantics and compulsory security protections. Ethos currently supports applications written in Go.

GReplay: A Programming Model for Kernel-Space GPU Applications

Xinya Zhang, Jin Zhao, and Xin Wang, Fudan University

The goal of this project is a GPU programming model for kernel-space applications with a high-level API, high portability, and low overhead. GReplay applications are developed with OpenCL and compiled in user space, but GPUs are invoked from a kernel module. The kernel-space component implements an abstraction layer over GPU drivers. Evaluation of four applications shows high throughput compared to Mesa 3D and Catalyst, an official driver.

Malleable Flow for Time-Bounded Replica Consistency Control

Yuqing Zhu and Jianmin Wang, Tsinghua University; Philip S. Yu, University of Illinois at Chicago

Replication schemes across datacenters typically trade off consistency with availability and operation latency. A system with malleable flow (M-flow) supports latency-bounded operations that maximize replica consistency within the given time. The key idea for malleable flow is to decompose the replication process into stoppable stages and then into a directed graph of ordered steps. Given a time constraint, the system will reform an execution flow into a path through the graph that guarantees fault-tolerance and maximizes consistency. An M-flow system has been implemented over Cassandra.

Rebasable File Systems for Enhanced Virtual Machine Management

Jinglei Ren, Bo Wang, Weichao Guo, Yongwei Wu, Kang Chen, and Weimin Zheng, Tsinghua University

Fast virtual machine cloning creates a VM by linking it to a base with block-level mapping. Two drawbacks of current approaches for fast cloning for VDI and cloud environments are that (1) updates to the base cannot propagate to derived images, and (2) derived images cannot seamlessly roll back to a previous base. Cinquain is a file-based storage that addresses these problems by providing a file system view for each VM. Cinquain can rebase operations for VMs by seamlessly changing the parent of a child view.

Experiences with Hardware Prototyping Solid-State Cache

Mohit Saxena and Michael M. Swift, University of Wisconsin—Madison

High-speed solid-state drives (SSDs) composed of NAND flash are often deployed as a block cache in front of high capacity disk storage. This work prototypes FlashTier, a lightweight, consistent and durable storage cache, on the OpenSSD evaluation board. To compensate for the low baseline performance of OpenSSD, the prototype implements a number of techniques within the OpenSSD device firmware to efficiently manage large flash block sizes and increased channel and plane parallelism. These techniques include: (1) merge buffer for aligned random writes, (2) read buffer for efficient random reads, (3) perfect page striping to maximize flash bank parallelism, and (4) minimized read-modify-write cycles for partial overwrites.

User-Mode Storage Systems for Storage-Class Memory

Haris Volos, Sankaralingam Panneerselvam, and Michael M. Swift, University of Wisconsin—Madison

Storage class memory (SCM) technology is byte-addressable, non-volatile, and has a low access time. This work redesigns the storage architecture of operating systems to take advantage of this new technology. The approach is to build a memory file system with high flexibility and performance by enabling direct access of SCM from user-mode applications. The system has three main components: (1) a user-mode library file system that implements naming and mapping, (2) a trusted file system service that enforces concurrency control and maintains the integrity of metadata, and (3) an SCM manager that securely records and enforces resource usage.

Distributed Systems and Networking

Summarized by Jim Cadden (jmcadden@bu.edu)

Spotting Code Optimizations in Data-Parallel Pipelines through PeriSCOPE

Zhenyu Guo, Microsoft Research Asia; Xuepeng Fan, Microsoft Research Asia and Huazhong University of Science and Technology; Rishan Chen, Microsoft Research Asia and Peking University; Jiaxing Zhang, Hucheng Zhou, and Sean McDirmid, Microsoft Research Asia; Chang Liu, Microsoft Research Asia and Shanghai Jiao Tong University; Wei Lin and Jingren Zhou, Microsoft Bing; Lidong Zhou, Microsoft Research Asia

Zhenyu Guo presented PeriScore, a procedural optimization technique for improving performance of data-parallel computation systems. This is achieved through pipeline-aware holistic code-optimization techniques.

Zhenyu began by defining network I/O as the bottleneck in distributed parallel pipeline jobs (such as MapReduce). One way to alleviate a network bottleneck is to reduce the data shuffling between computation procedures though optimizations. Traditional compilers do not optimize the procedure code in relation to the pipelining, and this can be a painstakingly process when done manually. PeriScore allows for automatic optimization of a distributed programs procedure code in the context of data flow.

By optimizing the procedure code directly, PeriScore removes unnecessary data, relocates operations, and calculates early predicates, which results in less data being shared across the network overall. The optimization process of PeriScore is to (1) construct an inter-procedural flow graph, (2) add safety constraints for skipping or shuffling code, and (3) transform code for the reduction of shuffling I/O.

In the question and answer session, Jason Flinn asked if the optimization based on static analysis can be improved with the addition of profiling. Zhenyu agreed that profiling would be sure to improve the overall performance increase as conservative approximations are currently done in cases where data sizes are unknown (e.g., streams).

MegaPipe: A New Programming Interface for Scalable Network I/O

Sangjin Han and Scott Marshall, University of California, Berkeley; Byung-Gon Chun, Yahoo! Research; Sylvia Ratnasamy, University of California, Berkeley

Sangjin Han presented MegaPipe, a new programming interface for scalable network I/O, designed to replace the standard BSD socket I/O. Sangjin began with the observation that message-oriented I/O (HTTP, RPC, key-value stores) with the BSD socket API can be very CPU intensive; small message sizes and short duration of connections lead to undesired performance, and adding additional cores does not alleviate the problem.

MegaPipe works through the combination of three system optimizations: (1) I/O batch processing assisted by a kernel library, (2) a per-core channel abstraction for a listening socket that allows for per-channel accept queues (avoiding contention), and (3) lightweight sockets that skip the file abstraction layers. MegaPipe assumes that file abstractions are no longer appropriate for sockets since sockets are short-lived and rarely shared.

In the evaluation section, MegaPipe was shown to improve throughput by up to 100% on an eight-core machine for messages of one kilobyte and smaller (smaller improvements were short for larger packet sizes). In addition, MegaPipe provided a near 15% throughput improvement for memcached and 75% throughput improvement for nginx for short connections. MegaPipe enjoys near linear scalability with evaluation run on up to 128 cores.

Mendel Rosenblum (Stanford) pointed out that people had just given up on using sockets and asked if their lightweight sockets are good enough to quit using other solutions. Sanjin answered that in most cases you don't need the full generality of sockets. When you really need the generality of sockets, you want to convert to our sockets. A researcher from MSR asked about the delay costs of the I/O batching. Sangjin responded that the batch size is small enough (~32 operations) so that it does not affect latency much. Ali Mashtizadeh (Stanford) asked if their research takes system scheduling into account. Sangjin explained that a basic assumption with MegaPipe is that there is one thread per core and that the MegaPipe process is scheduled like any other user-level application.

DJoin: Differentially Private Join Queries over Distributed Databases

Arjun Narayan and Andreas Haeberlen, University of Pennsylvania

Arjun Narayan presented DJoin, a technique for processing differentially private queries (including, but not limited to, 'JOIN') across distributed databases. Differential privacy is a process to control the amount of sensitive information that a database can release about its protected data set. Differential privacy software exists for central databases, as does non-private distributed join queries. DJoin applies both these techniques to allow for differentially private joins across distributed databases.

Arjun began with a motivating example of a scientist researching a recent outbreak of malaria in Albania. Ideally, this scientist would want to directly compare the records of who recently contracted malaria with those who had traveled to Albania (data that exists across databases of airlines and hospitals). However, free and open access to this information would be a giant violation of individual privacy and, in many cases, against the law.

Differential privacy works by factoring noise into the result of the queries made on a database. Every query on the database has a privacy cost attached, and the amount of noise added depends on the balance "spent" on that particular query. Once a user has spent their allotted resource they can no longer query the database. DJoin introduced two novel primitives: BN-PSI-CA, a differentially private form of private set intersection cardinality, and DCR, a multi-party combination operator that can aggregate noised cardinalities without compounding the individual noise terms.

In the evaluation section, DJoin was shown to incur non-trial computation costs, e.g., over an hour of computational overhead per query for databases greater than 30,000 rows. However, this work was described as "embarrassingly" scalable, shown by a nearly 4x increase through parallelizing across four cores. Arjun concluded by defining DJoin to be not fast enough for interactive use and more appropriate for offline analysis.

Being that this is a system's venue, it was no surprise that most questions involved curiosity surrounding the workings of differential privacy. Anunjun (Yale) asked if it is possible to support differentially private sum operations. Arjun replied that, yes, an extension of the existing count mechanism would be trivial. Henry Gibbs (Yale) inquired about a possible timing attack involving the size of the database and the time of the computation. Arjun explained that the set intersection is padded to the size of the entire database and noise is added to the polynomial size. Mike Freedman (Princeton) asked about the theoretical limitation of differential privacy, which requires a database to be "retired" after a certain amount of queries/cost have been processed. Arjun acknowledged that this characteristic of differential privacy was unfortunate and suggested that the lifespan of a sensitive database can be extended by carefully vetting access and queries.

Security

Summarized by Amit A. Levy (amit@amitlevy.com)

Improving Integer Security for Systems with KINT

Xi Wang and Haogang Chen, MIT CSAIL; Zhihao Jia, Tsinghua University IIIS; Nickolai Zeldovich and M. Frans Kaashoek, MIT CSAIL

Xi Wang from MIT presented several vulnerabilities resulting from integer overflow bugs. For example, such bugs allow an attacker to mount buffer overflow attacks or a malicious process to over-consume resources. Wang and his collaborators developed a static analysis tool, KINT, for identifying such bugs. They used KINT to find 114 bugs in the Linux kernel. They also propose two extensions to the C language and standard library that help mitigate integer overflow bugs.

Their case study yielded several interesting observations. First, of the 114 Linux kernel bugs they found, 79% caused buffer overflows or logical bugs. Second, two-thirds of the bugs had existing checks in the code, but the checks were wrong! Wang argued that this is evidence of how hard it is to reason about integer overflow bugs manually.

KINT is a tool for detecting integer overflows by statically analyzing LLVM IR (intermediate representation). KINT runs three separate passes on the program, one analyzing each function independently, another that performs whole-program analysis to reduce false positives from the first pass, and a third pass employing taint analysis using user annotations. Finally, KINT combines the results of all three passes and generates a bug report.

Wang discussed two mechanisms for mitigating the integer overflow bugs KINT finds. The authors added kmalloc_array, a new library call to the Linux kernel as of version 3.4. kmalloc_array implements the integer overflow check rather than relying on the caller to do so. Wang argued that while this fix targets a very specific bug, the bug is so common and potentially harmful that this simple library extension would have a large effect. The authors also propose adding the NaN value to the C language, which holds a special value for the results of computations that overflow an integer value. They implemented NaN in the clang C compiler.

Cristian Zamfir asked how Wang determined that the 125,172 possible bugs in KINT found in the Linux kernel were false positives. Wang responded that actually he's not sure whether they are false positives or actual bugs. Rather, the remaining 741 were bugs he was able to manually verify. Luis Pedrosa from USC asked how KINT deals with cases that are too hard for the solver to solve. Wang responded that they mark them explicitly as unsolved in the final report.

Dissent in Numbers: Making Strong Anonymity Scale

David Isaac Wolinsky, Henry Corrigan-Gibbs, and Bryan Ford, Yale University; Aaron Johnson, US Naval Research Laboratory

"Anonymity enables communication of sensitive ideas without fear of reprisal from government, organizations or peers." While, traditionally, anonymous systems must trade strong anonymity for scale, and users end up choosing weaker systems with more users, Daniel Wolinsky presented the argument that anonymous systems actually depend on large numbers of users—that weak anonymous systems with many numbers are often stronger than strong anonymous systems with few users. He introduced Dissent, a system that is able to provide strong anonymity while scaling to 1000s of active participants.

The key insight in Dissent is to combine the peer-to-peer architecture of DC-nets and Mix-nets to achieve strong anonymity and the client-server model of Tor to achieve scalability. For example, while each peer in DC-nets requires O(N^2) (N is the total number of participants) random number generations while in Dissent, clients need a random number generation per server, and each server needs one for each client—i.e., O(M*N) random number generations for the whole system. Similarly, communication cost in DC-nets is O(N^2) ciphertext transmissions, while Dissent uses multicast trees to achieve linear communication costs. Finally, Wolinsky presented an evaluation of Dissent that shows it can scale to thousands of participants.

Finally, Wolinsky presented an evaluation of Dissent that shows it can scale up to 1000s of participants. Mike Walfish from UT-Austin asked whether 2000 people are "really too many to throw in jail" since with Dissent it is very clear when someone is a member. He followed up, asking whether there was a way to hide membership. Wolinsky replied that the Tor project has made some progress in that regard, but that in the end he believes this comes down to an arms race with the adversary. Saurabh Bagchi (Purdue) asked whether there was a tradeoff between privacy and the number of honest servers that can be deployed. Wolinsky responded that the security of Dissent relies on the existence of at least one honest server as well as a number of honest clients.

Efficient Patch-Based Auditing for Web Application Vulnerabilities

Taesoo Kim, Ramesh Chandra, and Nickolai Zeldovich, MIT CSAIL

Buggy Web applications can lead to security vulnerabilities that may only be discovered long after the applications have been introduced. If a vulnerability has been around for months or years, has it been exploited? By whom? Taesoo Kim presented a technique for using security patches to identify past attacks.

The key insight is that since security patches render previous attacks harmless, comparing the execution of each past request with the patch and without it will expose differences. For example, the same request may result in different SQL queries to the database with and without the patch. If the request in fact behaves differently, that request may be part of an attack. However, a naive implementation would execute each request twice—so auditing one month of traffic would require two months.

Kim identifies three opportunities he and his colleagues used to improve performance in their auditing system. First, a patch may not affect all requests, so not all requests need to be re-executed. They used control-flow filtering to determine which requests could not be affected by the patch. Second, much of the pre-patch and post-patch execution runs are identical, so there is no need to run both, identical, versions. Finally, multiple requests may execute similar code, so further redundancies can be eliminated. The authors memoized requests throughout the re-execution to avoid re-running similar requests.

Using all of these techniques, the authors were able to audit requests 12–51 times faster than the original execution. They evaluated the effectiveness of their system on patched vulnerabilities from MediaWiki (using Wikipedia traces) and HotCRP (using synthetic workloads). Finally, they found that the overhead of their system during normal execution was about 14% higher latency and 15% lower throughput.

Matvey Arye (Princeton) asked about the overhead of storing all of the information along with requests. Kim replied that in their Wikipedia traces the overhead averaged 5.4 KB per request. Jonas Wagner (EPFL) asked how they can re-execute requests without access to state that might only be available, such as specific data, in the production database. Kim clarified that instead of making actual database calls, their system records responses from external sources like the database and replays those responses during re-execution.

Potpourri

Summarized by Amit A. Levy (amit@amitlevy.com)

Experiences from a Decade of TinyOS Development

Philip Levis, Stanford University

Phil Levis looked back at over a decade of the development and community surrounding TinyOS. TinyOS began as a research project in 1999 at UC Berkeley aimed at building an operating system for sensor networks composed of wirelessly connected low-power devices. Today, TinyOS boasts over 25,000 downloads every year and is installed on hundreds of thousands of nodes. It has spawned hundreds of research papers and garnered a worldwide community.

Levis reflects on two design principles that emerged from TinyOS: (1) minimizing resource use and (2) structuring interfaces and code to avoid bugs. He recaps static virtualization, an important technical result of the project that virtualizes resources at compile time, which allowed TinyOS to achieve the two design principles. Finally, he shared his opinion of how the community around TinyOS grew, how the nature of the community affected the design and development, and what could have been done differently to keep expanding the community of users.

The TinyOS community developed what Levis calls the "island syndrome." In particular, the community gained a critical mass of developers that was focused on making it easier to build complex applications rather than ensuring that simple applications are easy to build. The "island syndrome" develops when a developer community is large enough that it no longer needs to grow, and allows itself to be inward rather than outward looking. In his talk, Levis discussed the implications this had on TinyOS and how it may have contributed to projects such as Arduino surpassing TinyOS in popularity.

Marc Chiarini (Harvard) wondered whether it would be difficult to re-architect TinyOS. Levis responded that it wouldn't be that difficult as there is not much code. However, Contiki has filled that role, and is much more accessible. Aaron Carroll (NICTA/UNSW) said that having lots of users is a good thing, as building specialized software is hard, and wondered whether making it easy to use is the right metric. Levis said that this is not true in all cases, but if he had to do it over, he would still pick doing the hard things so that researchers didn't have to. Matthew Fernandez (NICTA/UNSW) asked whether making things simpler and easier in the interest of improving usability for novices might have prevented reaching research goals. Levis replied that no, they wanted to make it easy to teach by keeping things simple. Experts can still use the system as they want. Eric Sedlar (Oracle) had worked on Sunspot, and compared research goals to usability. Levis said that the Sunspot goal for usability was much higher than TinyOS's. Someone complained about the lack of tools for developing TinyOS, and Levis agreed that tools could have made things easier. They assume a knowledge of C and event style programming.

Automated Concurrency-Bug Fixing

Guoliang Jin, Wei Zhang, Dongdong Deng, Ben Liblit, and Shan Lu, University of Wisconsin—Madison

Guoliang Jin presented CFix, a system for fixing concurrency bugs automatically. The premise of the work is that there has been much work on finding bugs, but that only addresses part of the problem—bugs must be fixed as well. Even if bugs are found automatically, fixing them requires a lot of human effort, which itself is error prone. Automatic bug fixing can address this problem on accounts. However, automatically fixing bugs requires knowing both the desired behavior of a program and how to achieve this behavior.

CFix specifically addresses fixing synchronizations in concurrent, multithreaded programs. Synchronization bugs are an attractive target because they are an acute problem, but also because the problem is tractable. In particular, correct program behavior is easily observable in all but the (rare) buggy interleaving. Moreover, achieving correct behavior can be done using synchronization operations to disable the buggy interleavings.

Jin gave an overview of CFix and discussed some of the challenges he and his collaborators faced and addressed in building CFix. CFix is a six-step system that uses existing bug finding tools along with a set of novel techniques to find and repair concurrency bugs in programs.

Session chair Florentina Popovic (Google) asked Jin if he had suggestions for tool writers. Jin said they the picked four bug detectors, but none included false-positive detection. Chris Rossbach (MSR) asked if condition variables are allocated statically or dynamically. Jin answered that they are allocated statically.

All About Eve: Execute-Verify Replication for Multi-Core Servers

Manos Kapritsos and Yang Wang, University of Texas at Austin; Vivien Quema, Grenoble INP; Allen Clement, MPI-SWS; Lorenzo Alvisi and Mike Dahlin, University of Texas at Austin

Traditionally, there has been no efficient way to build services which are both dependable and high-performance. Specifically, dependability is normally achieved through replicated state machines, and high-performance is achieved with concurrency. But replicated state machines cannot leverage concurrency since they service all requests sequentially; replicated state machines rely on a deterministic ordering of requests.

Manos Kapritsos introduced Eve, an architecture that, he argued, eliminates the dependability versus performance tradeoff in such services. Eve achieves this by reversing the architecture of replicated state machines. Instead of replicas agreeing on the order of requests and applying those requests sequentially, Eve allows replicas to apply requests concurrently and agree on their resulting states. The key insight that enables this is that commutative operations can leverage parallelism so that only conflicting operations must be ordered consistently.

Eve uses a Mixer to group requests that are estimated to be commutative. Once such a group is serviced by the replicas they verify the resulting state with other replicas. If the states agree, replicas proceed in servicing other requests. Otherwise, the states are rolled back and Eve falls back to sequential, ordered execution. The authors evaluated Eve using the H2 database engine. They found that on a Web workload, Eve performed only 15% slower than an unreplicated server (the theoretical upper-bound) and 6.5 times faster than a traditional replicated state machine.

Replication

Summarized by Edmund Wong (elwong@cs.utexas.edu)

Spanner: Google's Globally-Distributed Database

James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford, Google, Inc.

Awarded Jay Lapreau Best Paper!

Wilson Hsieh presented Spanner, a distributed high-performance multiversion database that supports ACID, schematized tables, a semi-relational data model, and a SQL query language. More importantly, Spanner allows data to be replicated across multiple datacenters and sharded across thousands of machines in each datacenter, while ensuring external consistency (i.e., linearizability) for distributed transactions. The commit order of transactions is the same order that users see the transactions executing. This rich interface makes Spanner easier for programmers at Google to build applications, as compared to a key-value store.

The key enabling mechanism that allows Spanner to provide external consistency is TrueTime, which provides extremely precise, consistent global wall-clock time with bounded uncertainty. To use TrueTime, a node consults local and remote GPSes and atomic clocks (timemasters) located across various datacenters. Marzullo's algorithm is then applied to the reported times to compute the estimated current time and the maximum uncertainty in this estimate. TrueTime takes into account local-clock offset as well as worst-case local-clock drift, which is conservatively estimated.

Spanner uses this timestamp to designate when transactions occur, effectively transforming the problem of determining the order in which transactions commit into a problem of assigning accurate timestamps that respect wall-clock time. For read/write transactions, Spanner uses strict two-phase locking to acquire locks for all operations. When this transaction is to be committed, Spanner uses two-phase commit and additionally uses TrueTime to assign a single timestamp for all the operations in the transaction. Because Spanner does not actually know whether the clock it observed is accurate, it always chooses to be safe by selecting a timestamp that is the latest possible current time (the current estimated time plus the highest reported uncertainty). Spanner also ensures that the commit operation does not occur until the current time, with the maximum reported uncertainty, is known to be later than this timestamp. Although the average delay of committing operations in this fashion is double the maximum reported uncertainty, TrueTime's tight bounds on uncertainty makes this delay small (about 10 ms in the common case) and allows the delay to be overlapped with real work. TrueTime also enables lock-free read transactions; because write operations are only visible after their associated timestamps have passed, Spanner can ensure external consistency by delaying read operations at a replica until the replica has seen the appropriate writes (those that have happened before the read operation's timestamp).

Hsieh stated that Spanner is already running in production, storing Google's ad data, and has replaced a sharded MySQL database. Hsieh showed that, in a given year, clock failure, which could result in external consistency violations, was empirically rare: during this year, CPUs were six times more likely to fail than clocks. Hsieh also showed that during an evaluation period, TrueTime reported close to zero uncertainty more than 90% of the time, and less than 10 ms of uncertainty 99.9% of the time. Hsieh believes that in the future, better time mechanisms can be implemented to make TrueTime even more precise. Furthermore, Hsieh said that Google is planning on building out database features, including finishing the implementation of basic features and efficiently supporting rich queries.

During the active Q&A, a number of audience members asked variations on the question, what happens if TrueTime is inaccurate? Hsieh replied that external consistency will be violated. Why were specific timestamps used for transactions instead of timestamp ranges? It was for simplicity. What happens when some datacenters are located in regions with bad GPS coverage? This increased uncertainty in TrueTime would result in Spanner slowing down for datacenters with higher uncertainty. One audience member pointed out an anomaly in a chart that showed time uncertainty increased for a period of time. Hsieh explained that the local timemaster for TrueTime was down during that period. Fritz Nolan asked what happens while servers are deciding on the timestamp. Hsieh replied that operations simply appear to have occurred at that specific timestamp.

Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary

Cheng Li, Max Planck Institute for Software Systems; Daniel Porto, CITI/Universidade Nova de Lisboa and Max Planck Institute for Software Systems; Allen Clement, Max Planck Institute for Software Systems; Johannes Gehrke, Cornell University; Nuno Preguiça and Rodrigo Rodrigues, CITI/Universidade Nova de Lisboa

Geo-replicated services traditionally choose between strong consistency, which provides natural semantics at the cost of high latency, and eventual consistency, which provides low latency but can result in undesirable behaviors. Cheng Li presented RedBlue consistency, a hybrid consistency model that enables geo-replicated systems to achieve strong consistency only on operations that require such guarantees while preserving better performance on those operations that do not. Developers differentiate these operations by labeling those that require strong consistency as red; operations that require only eventual consistency are labeled blue. Based on this labeling, all operations are causally ordered, but red operations are totally ordered with respect to each other.

Because RedBlue consistency's performance depends on operations being blue, Li described a novel technique to increase operations commutativity: divide operations into two sub-operations, one which calculates the side effect of the red operation without applying said operation (e.g., the amount of interest that needs to be added to a user's account in a banking application), and the other which applies the previously calculated side effect (e.g., add the previously calculated interest). By breaking a red operation into these two sub-operations, these sub-operations can often be labeled as blue as long as they do not violate the application's invariants.

To evaluate RedBlue consistency, Li et al. implemented a replication and coordination layer called Gemini, which supports RedBlue consistency and consists of a coordinator and a storage engine at every geo-replicated site. When a blue operation is received at a site, the operation is accepted immediately and replicated to the remote site by the corresponding local coordinator. For a red operation, the coordinators communicate in order to ensure that they agree on the order of this operation. Li et al. evaluated the efficacy of RedBlue consistency by running the TPC-W and RUBiS benchmarks as well as a social networking app (Quoddy) atop Gemini. By breaking up operations into two commutable sub-operations, Gemini was able to label a vast majority of operations as blue. Moreover, Li showed that RedBlue consistency improves both user-observed latency and peak throughput of TPC-W in various multi-site deployments.

Yang Tang (Columbia) asked whether commutative sub-operations could be automatically created from operations and someone from Cornell asked whether Gemini required users to determine commutativity of operations. Li replied that both tasks are done manually and, from his experience, were not difficult. However, automating this process is planned as future work. Mike Freedman (Princeton) asked whether red operations may end up blocking blue operations at a client if the red operations have to be totally ordered. Li answered that blue operations are never blocked by red operations. Finally, someone referred Li to some relevant work, namely, generalized Paxos and generalized broadcast.

OSDI Poster Session 2

Summarized by Feng Yan (fyan@cs.wm.edu)

Application-Aware Stateful Data Routing for Cluster Deduplication in Big Data Protection

Yinjin Fu, National University of Defense Technology and University of Nebraska-Lincoln; Hong Jiang, University of Nebraska-Lincoln; Nong Xiao, National University of Defense Technology Contact: yinjinfu@gmail.com

Yinjin et al. presented a data routing scheme for cluster deduplication in which they first exploit application awareness to divide a backup data set into many small application-affined subsets with negligible cross-subset data overlap. They then assign each application-affined subset to a group of deduplication server nodes by an application-aware routing algorithm. Within each node group, data of an application-affined subset is routed to individual target nodes by leveraging data similarity. By combining application-aware routing among node groups and similarity-based routing within each group, their data routing scheme is able to significantly mitigate the challenge of a deduplication node information island.

High Performance Modular Packet Processing with Click and GPU

Weibin Sun and Robert Ricci, University of Utah

In order to build a flexible high performance packet processing system that could run at a 10 Gbps line rate, Weibin Sun and Robert Ricci used the modularized Click router and integrated GPU computing for computation-intensive processing. But Click was not originally designed for user-mode zero-copy packet I/O and parallel GPU computing, so they adopted a series of technologies to deal with obstacles in Click. They implement Click-level improvements, including GPU-related elements and the Netmap/CUDA/Click integration for zero-copy memory sharing.

Granary: Comprehensive Kernel Module Instrumentation

Peter Goodman, Angela Demke Brown, Akshay Kumar, and Ashvin Goel, University of Toronto

Granary works on multicore processors with preemptive kernels, and incurs a modest decrease in throughput of 10% to 50% for network device drivers. Granary has been used to isolate and comprehensively instrument several network device drivers (e1000, e1000e, ixgbe, tg3) and file system modules (ext2, ext3). Granary also has been used to develop an application which enforces partial control-flow integrity policies. These policies disallow modules from executing dangerous control-flow transfers.

Improving MapReduce Performance in Highly Distributed Environments using End-to-End Optimization

Benjamin Heintz and Abhishek Chandra, University of Minnesota; Ramesh K. Sitaraman, University of Massachusetts, Amherst

The MapReduce assumption that the data and compute resources are available in a single central location no longer holds for many emerging applications, where the data is generated in a geographically distributed manner. The authors developed a modeling framework to capture MapReduce execution in a highly distributed environment comprising distributed data sources and distributed computational resources. This framework is flexible enough to capture several design choices and performance optimizations for MapReduce execution. They modified Hadoop to follow their model-driven optimization, and experimental results show their approach outperforms Hadoop's existing dynamic load-balancing mechanisms.

Towards Decentralized Memory Management in a Multikernel

Simon Gerber and Timothy Roscoe, ETH Zurich

High synchronization overhead and contention on shared data structures prevent mainstream OSes from adapting memory-management to heterogeneous and multicore systems. Simon Gerber and Timothy Roscoe tried to build a memory management system using explicit communication, distributed systems techniques, and a share-nothing architecture and treating caches and main memory as distinct units. One goal of that work is to make a single memory management system on a heterogeneous machine a possibility.

Model Checking Mobile User Interface Constraints

Kyungmin Lee and Jason Flinn, University of Michigan; T.J. Giuli, Ford Motor Company; Brian Noble, University of Michigan; Christopher Peplin, Ford Motor Company

Vehicular UI has the no visual and cognitive distractions to the drive constraints. In order to find violations of best practice design guidelines in the application, manual testing is needed, which is too time-consuming and difficult. Kyungmin Lee et al. use model checking to explore the application automatically and find UI violations by reporting to user via annotated screenshots.

OSDI Posters 2

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

Serval: An End-Host Stack for Service-Centric Networking

Erik Nördstrom, David Shue, Robert Kiefer, Prem Gopalan, Matvey Arye, Jennifer Rexford, and Michael J. Freedman, Princeton University

Serval offers a Service Access Layer (SAL) located between the network and transport layers of a traditional network stack. This layer uses service and flow identifiers, decoupling host identification, flow multiplexing, and routing.

Serval enables load balancing (a group of processes shares a service ID), host mobility, and virtual machine migration (a flow is identified by a flow ID—the underlying IP address can change), and late binding. It does not modify the network layer, keeping the benefits of IP's hierarchical routing.

PacketFlow: Managing Network Energy in Smartphones

Arun Bharadwaj, Evgeny Vinnik, Arrvindh Shriraman, and Alexandra Fedorova, Simon Fraser University

PacketFlow minimizes a smartphone's energy usage by putting the network into low-power mode as often as possible. It slightly delays some network packets so they can be sent in bursts. In-between bursts, the network is put in low-power mode. PacketFlow uses per-application queues and distinguishes between latency-sensitive and elastic applications. Only packets from elastic apps are delayed in order to minimize the user-perceived latency.

Formalization of a Component Platform

Matthew Fernandez, Ihor Kuz, and Gerwin Klein, NICTA and University of New South Wales

Matthew Fernandez et al. aim to scale formal verification to large systems using a component-based approach. By formally specifying what components are and how the glue code between the components must behave, and by implementing the system on top of the verified kernel seL4, the component platform becomes the first of its kind able to provide correctness guarantees. Currently, formal specifications for component concepts have been completed.

Evaluating Software Managed Memory with MapReduce

Craig Mustard, Alexandra Fedorova, and Arrvindh Shriraman, Simon Fraser University

Craig Mustard proposed a variant of MapReduce that runs efficiently on architectures where no coherent cache is available but a small amount of high-speed L1 memory exists. The key is a special data structure used to group results by key before the reduce phase. It is a linked list of keys stored in small blocks (usually 1 KB) that fit in L1 memory. Pointers in this list are relative to the block, so blocks are relocatable and can be swapped in and out of L1 memory as needed. Using blocks results in an 11x speedup over vanilla MapReduce, and a 31% overhead with respect to an idealized system where a hardware cache is made available.

Moscilloscope: High-Rate Power Data Acquisition on Mobile Systems

Jun Zhou and Tao Peng, Texas A&M University; Mian Dong and Gary Xu, Samsung Telecommunications America

Moscilloscope is a framework, including a specialized hardware sensor and an associated software stack, that enables a smartphone to measure the real-time power consumption of itself at a high sampling rate (200 Hz). Jun Zhou and Mian Dong presented a prototype that consists of a PandaBoard running Android, an external battery pack, and a MAXIM DS2756 EVM sensing the power consumption of the PandaBoard at the battery terminals. A user-space app showed the power consumption of the PandaBoard in real time. They envision that Moscilloscope, as an enabling technology, will be useful for many applications such as power profiling/optimization, energy accounting/management, application debugging/verification, malware detection, and even user authentication.

Naiad: Decentralized Low-latency Incremental and Iterative Computation

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

Naiad is a successor to Dryad, a system for data-parallel computations. It adds the ability to incrementally recompute results as the input changes, despite the challenge that many data-parallel algorithms are iterative. Naiad achieves this by expressing iterations as fixed-point computations, and by keeping a trace of the computation history. At every new iteration, or when new input data arrives, only the affected nodes are updated. Naiad has been implemented on top of .NET's LINQ framework, and a demo was presented that computed strongly connected components in a Twitter graph with sub-second latency.

Testing & Debugging

Summarized by W. Michael Petullo (mike@flyn.org)

SymDrive: Testing Drivers without Devices

Matthew J. Renzelmann, Asim Kadav, and Michael M. Swift, University of Wisconsin—Madison

Developing device drivers is difficult; even testing them demands a tremendous effort. Matthew Renzelmann presented SymDrive, a system that simplifies testing device drivers. Matthew began his talk describing some of the things that make testing device drivers unique. Perhaps most obviously, a single driver often supports a collection of devices; testing on all supported devices incurs costs in both money and time. Matthew explained that SymDrive supports both Linux and FreeBSD. Linux in particular is known for a high rate of change in its tightly coupled code. Thus it is likely that an individual driver will often be affected by changes in the kernel proper. Drivers must be tested not only when their code changes, but also often following changes elsewhere in the kernel. Matthew remarked that his team found dozens of kernel patches noted as "compile tested only." It is not only that testing is hard; with any complex piece of software, testing is inherently incomplete. A tool that automates testing or helps make it more comprehensive is always welcome.

Matthew went on to describe SymDrive and explain its goals. His team is interested in finding deep, meaningful bugs, including those that span multiple driver invocations. They also want it accessible to developers and applicable to any class of device drivers. To meet these goals, SymDrive is based on the S^2E symbolic execution engine. The system uses a combination of explicit pre-/post-conditions, checking of runtime properties, a test framework, and a support library to meet the project's goals. [EDITORS: S2E appears just as I have it on its originating site: http://dslab.epfl.ch/proj/s2e; it can be typeset as S [squared] E as well. $S^{2}E$ is LaTeX for S[squared]E RIK]

Using SymDrive, Matthew's team found 39 bugs in 26 drivers. Several of these bugs have been validated by the relevant kernel developers, and 17 bugs have been fixed. Matthew presented an argument that the burden on testers remained low. To further demonstrate the usefulness of the tool, Matthew noted that most drivers take around 30 minutes to test; thus driver developers could probably use SymDrive to test at a rate appropriate for the changes being committed to the kernel proper.

Philip Levis (Stanford) asked about path explosion, whether they test all the ways that interrupts may occur. Matthew noted that SymDrive does not test all possible interleavings of interrupt handling; this is natural because testing cannot be complete, and it does not mean that SymDrive is not a useful tool. In fact, SymDrive minimizes false positives. Likewise, he argued that breaking out of loops early and favoring successful paths are appropriate design choices.

Be Conservative: Enhancing Failure Diagnosis with Proactive Logging

Ding Yuan, University of Illinois at Urbana-Champaign and University of California, San Diego; Soyeon Park, Peng Huang, Yang Liu, Michael M. Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage, University of California, San Diego

Ding Yuan began his talk by stating that production errors are often difficult for engineers to reproduce. The are a few reasons for this, most notably that privacy concerns often make it hard for engineers to gather the failure-triggering input, and it might be expensive to reproduce the precise environment used by each customer requiring support. Most likely, engineers rely on customer-provided logs as they attempt to provide troubleshooting support.

How useful are the logs provided by current software? By sampling open source bug reports, Ding noted that less than 1/2 of failures resulted in a useful log message. This includes large, mature projects such as Apache and PostgreSQL. This statistic begs the question, "How difficult would it be to add useful logging to the remaining conditions?" Ding's team classified seven patterns, two of which he presented in his talk, that indicate it is feasible to add logging. Of the projects he studied, 77% were candidates for an easily-added log function.

Ding talked about some more subtle pitfalls. For example, it appears that developers who handle errors in their code often think that this supplants the need for logging. Not so—error handling itself is sometimes buggy, and errors are often useful for troubleshooting even when handled.

Ding's team produced a tool named Errlog to help automate adding log code. This helps break the cycle of adding log code only after a user reports a problem that would have been easier to isolate had the logging function been present in the first place. Their evaluation included a performance review, effectiveness statistics, and a user study.

Someone asked if turning on verbosity helps with seeing more error messages. Ding replied that verbosity is not enabled in production as it adds up to 90% overhead. ErrLog adds more error messages, not verbosity. Carl Waldspurger asked if they could do this without source code, using something like library call interposition. Ding said it was possible if symbol information is present, but not possible to tell if error checking was already in the code. Margo Seltzer said that one takeaway is that we need to teach programmers to create more error messages. Geoff Kuenning followed up by saying there are macros that can be used to make error log creation easier. Ding concluded by saying that when programmers are under deadline, adding error messages does not add to productivity. But IDEs could be used to automatically add error logging and prompt programmers to fill in useful descriptions of errors.

Jay Lapreau Best Student Paper X-ray: Automating Root-Cause Diagnosis of Performance Anomalies in Production Software

Mona Attariyan, University of Michigan and Google; Michael Chow and Jason Flinn, University of Michigan

Awarded Jay Lapreau Best Student Paper!

Michael Chow and team want to make it easier to find the source of performance problems. Chow began by introducing the audience to an anecdote involving Postfix, a popular mail transfer agent. A user was finding that sending email through Postfix from a single email account was very slow, even though other senders performed well. The user was experienced, and he tried to diagnose the problem using top, iotop, and wireshark. He then viewed Postfix's logs and noticed a high volume of messages. But why was this? Postfix has many configuration options; it turned out that one had caused the verbose logging of this particular account. Michael noted that the answer, once found, is obvious. However, until then the situation is like "finding a needle in a haystack."

Clearly, identifying the source of bad performance is hard. X-ray attempts to attribute performance costs to their root causes. It performs effectively having found the correct cause of the majority of performance problems when presented with scenarios based on Apache, Postfix, PostgreSQL, and lighttpd. The online overhead of the tool is small, and analysis (performed offline) would likely often beat the time taken to manually troubleshoot.

Marc Chiarini asked whether bugs could mask other bugs. Michael replied that they provide a ranked list of possible root causes. Ryan Stutsman (Stanford) asked whether X-ray can work with multi-component systems. Michael said you can run X-ray on multiple machines. Rik Farrow asked whether sysadmins would be able to install PIN. Michael said that installing PIN is the easiest part, as running X-ray includes running a modified kernel that is deterministic, and that is the greater burden.

Isolation

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

Pasture: Secure Offline Data Access Using Commodity Trusted Hardware

Ramakrishna Kotla and Tom Rodeheffer, Microsoft Research; Indrajit Roy, HP Labs; Patrick Stuedi, IBM Research; Benjamin Wester, Facebook

Ramakrishna Kotla discussed the inherent tradeoff that exists between mobility and security—especially when the client is untrusted. In thin-client architectures, data is stored at the server and accessed only when a client requests it. This model is good for security, because sensitive data is protected by the server, but thin clients have limited mobility in the event of disconnection. Rich-client architectures push data to the client, which is great for mobility, but offers poor security when clients cannot be trusted.

Ramakrishna posed two questions. First, he asked how a system can ensure that a user has not accessed data in the past, even when the user, the applications, the OS, and the hypervisor are untrusted. He then questioned how a system can prevent future accesses to data, even when an untrusted client has access to the data's encryption keys and physical storage. Ramakrishna introduced Pasture, a messaging and logging library that operates in three phases. In the first phase, a sender and receiver exchange keys and data. In the second phase, the receiver may access and record data even when offline. In the final phase, the sender can audit the client device and determine whether the data was accessed or not. At the end of the audit, the server may also receive guarantees that the data is inaccessible to the user in the future.

Ramakrishna explained that Pasture provides two properties, the first of which is access undeniability. Access undeniability guarantees that a user who obtains access to data sent by a correct sender cannot lie about that access without failing an audit. The second property, verifiable revocation, allows a user to revoke access to data and generate a verifiable proof of revocation, which guarantees that the user did not and cannot ever access the data in the future. Pasture provides these properties by leveraging commodity trusted hardware, namely Trusted Platform Modules (TPM) and secure execution mode (SEM) extensions, found on most modern devices.

When asked about crashes during the checkpoint routine, Ramakrishna explained that Pasture does not guarantee liveness. The system would not allow the user to continue—keys would be unusable and application-level recovery is assumed. Rik Farrow pointed out that Pasture leverages commodity hardware on Intel platforms, but that most mobile devices are ARM based. Ramakrishna noted that ARM provides trusted zones, but that this is something they have yet to explore. Kevin Walsh (College of the Holy Cross) asked why Pasture does not use trusted execution to provide more interesting predicates for access control policies. Ramakrishna replied that the approach would be vulnerable to snoop attacks.

Dune: Safe User-Level Access to Privileged CPU Features

Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazières, and Christos Kozyrakis, Stanford University

Adam Belay asserted that privileged CPU features are powerful—they are fundamental to the functioning of operating system kernels. Yet privileged CPU features have additional compelling uses. For example, garbage collection performance would be greatly improved with direct access to page tables. However, available approaches all have drawbacks. A kernel module could be written to provide access to page table information directly, but modifying the kernel for individual applications quickly becomes intractable. Microkernels throw out the entire kernel stack, and virtual machines are strictly partitioned. Adam presented Dune as an elegant solution.

Dune provides safe user-level access to privileged CPU features. Dune does not require kernel modification—if a feature is needed by an application, that feature is simply used by the user-level application. Further, the process is still a traditional process in every other way. Dune provides these features by leveraging existing virtualization hardware, namely Intel VT-x. Dune provides distinct privilege modes, virtual memory, TLB management, and exceptions through the open source libDune library and the Dune kernel module.

Adam next explained Dune's inner workings. The CPU is broken into two orthogonal operating modes. Host mode, normally used for hypervisors, is where the kernel runs. Guest mode, normally used by the guest kernel, is where ordinary processes run and are given safe access to privileged CPU features. Memory management leverages two layers of page translation. The extended page table (EPT) maintains safety and correctness, and is configured to support sparse address spaces. In Dune, VMCALL is used to perform normal Linux system calls, since SYSCALL traps back to the process. Finally, signals are injected as interrupts, which automatically forces a transition into ring 0, at which point Dune can correctly service the signal.

Alex Wolman (MSR) asked about exposing I/O devices with virtualization support in Dune. While this wasn't evaluated in the paper, Adam believes it is possible. Intel provides VT-d, which can make it relatively easy to expose devices directly to user programs, and SR-IOV permits them to be partitioned into multiple instances so that more than one Dune process could potentially use the same device safely. Matthew Fernandez (NICTA/UNSW) asked whether there were plans to virtualize VMCS data structures and other extensions needed to nest Dune. Adam noted that the Dune philosophy is to only provide hardware features that can be supported without emulation. However, the reverse situation, where Dune is run within a hypervisor, is likely possible, a belief which is supported by the OSDI '10 best paper, "The Turtles Project," in which the VT-x instructions are virtualized.

Performance Isolation and Fairness for Multi-Tenant Cloud Storage

David Shue and Michael J. Freedman, Princeton University; Anees Shaikh, IBM T.J. Watson Research Center

David Shue explained that for key-value stores that support basic get and set operations, that have asynchronous durability, and that operate in an environment with full bisectional bandwidth, achieving predictable performance in the face of resource contention is hard. Fair queuing works well in monolithic systems, but not for services designed as distributed systems. He suggested that tenants really do not care about their resource allocation on particular nodes, but that they would rather treat the entire system as a black box, and receive guarantees in terms of system-wide aggregate resources. A system that provides such guarantees should not treat rates as hard limits either, because hard limits could leave the system underutilized. Instead, rates should be conveyed as proportions of total system resources, and each tenant given a max-min fair share.

David identified and decomposed three mechanisms currently employed by key-value storage systems. The partition placement mechanism partitions data and assigns data partitions to specific storage nodes; request routers direct tenant requests to the nodes housing the desired partition, and replica selection is the mechanism they use for load balancing read requests across replicas; and the queuing mechanism mediates requests once they arrive at individual storage nodes. However, since the goal of Pisces is to provide per-tenant weighted fair shares of aggregate system resources, Pisces provides a fourth weight allocation mechanism to translate global weights down to weights at local nodes. Weight allocation is necessary because Pisces makes no assumptions about object popularity, and for skewed popularity, tenants should be given resources where they need it most.

To prevent hot partitions from different tenants colliding on the same node and exceeding the node's throughput capacity, Pisces places partitions according to per-partition demand. A centralized controller collects per-partition statistics and runs a bin-packing algorithm to fit the partitions within node capacity limits. Local tenant weights are allocated according to tenant demand, and the centralized controller collects per-tenant demand statistics to identify demand-weight mismatches. It alleviates mismatches by shifting weight from a tenant with excess weight to an under-provisioned tenant, and then reciprocating the weight swap to preserve global weight proportions. Request routers implicitly detect weight mismatches using request latency as a proxy and adjust policy to match local weights. Nodes uses dominant resource fair queuing locally; each node tracks a per-tenant resource vector and periodically determines each tenant's dominant resource. In this way, the node gives each tenant equal shares of the resource it wants most.

Someone from VMware asked how one could determine system capacity without making assumptions about request types and request distribution. David noted that different workloads would change the overall system capacity, and they currently do not have a solution. However, Pisces assumes asynchronous durability; since it is not very concerned with disk I/O, and because it deals mostly with network I/O effects and simple requests that don't consume variable amounts of CPU, calculating system capacity is made a little bit easier. David was then asked how Pisces handles the key-value cache space on the server. He replied that that was another part of the resource spectrum that Pisces does not explicitly deal with. The system was built on Membase, and the amount of memory that each tenant gets is provisioned up front. He notes that performance will depend on the working set size of each tenant.