2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10)

June 22, 2010
Boston, MA

Foreword by Alva L. Couch (couch@eecs.tufts.edu)

For those of us who attended both HotCloud '09 and '10, there has already been a transformation. In 2009, the workshop was dominated by definitions and struggles, including carefully defining the kinds of clouds and struggling with the oxymoron we call "cloud security." In 2010, surprisingly, the community has made some peace with the invariant properties of clouds, and the workshop was overwhelmingly solution-centered, with many small problems addressed and at least a hint that the grand-challenge problems of cloud privacy, security, isolation, and predictability may have technical solutions. More profoundly, ideas have matured, and it is reasonable to expect that many of the papers presented in 2010 will appear shortly in full paper venues. As scary as cloud security and privacy issues may be in isolation, clouds still have the potential to transformationally improve security and privacy of data for many small enterprises, whose non-cloud security practices are currently inadequate. In short, while 2009 painted a fairly bleak picture with rather stormy clouds, 2010 looked forward to a relatively sunny future, with a few cirrus clouds unobtrusively floating high in the sky.

PERFORMANCE AND POWER

Summarized by Joshua Reich (reich@cs.columbia.edu)

- ***Seawall: Performance Isolation for Cloud Datacenter Networks***
  Alan Shieh, Cornell University and Microsoft Research; Srikanth Kandula, Albert Greenberg, and Changhoon Kim, Microsoft Research

Currently, applications can't guarantee performance after migrating to the cloud. Globally shared resources, such as network capacity share, are probably the most difficult to control, and this is when other applications running in the cloud are well behaved. When malicious users are present (e.g., DNS attacks from cloud tenants) this problem becomes even worse.

Alan Shieh presented Seawall, whose goals are to isolate tenants, controlling their share of the network, while still utilizing network capacity effectively. Additional constraints are that tenant code is untrusted and system churn should be minimized as VMs come and leave. Possible solutions include counter-based flow level AC (can't handle sufficient number of flows), QCN (not yet standard, requires network upgrades, solves a somewhat different problem), and end-to-end bandwidth restrictions (easy to subvert).

Seawall works by having all traffic flow through a "Seawall port" on the hypervisor. It requires no central controller (each host runs its own rate controller); is enforced by "Seawall ports," sitting on the packet forwarding path (one per

source-dest pair); and detects congestion on src-dest paths, using direct feedback, AIMD.

Practically, this approach needs to keep overhead low—in particular, the main bottleneck is the Seawall port, which interacts with every packet. Encapsulation might be one approach, but this adds lots of overhead and breaks header format optimizations in switches and hash-based load balancing. The authors suggest the possibility of "bit stealing," using spare bits from existing headers—e.g., IP-ID field and some other bits from the constant part of the Seq#—and although this seems like a bit of a kludge, they claim it produces good performance in their simulator validation experiments.

Someone asked about the Amazon security group, and Shieh answered that Seawall can add that directly to the Seawall port. Another person suggested speculative network capacity use (e.g., spot pricing). Shieh noted that they haven't addressed this directly, although it doesn't appear that this would be a difficult extension. Someone asked about the CPU time required for Seawall ports, and Shieh said they spent a lot of time optimizing the code, but 40% of the core cycles are still needed to implement this. Centralized rate limiting? They have considered other distributed feedback loops but believe centralized rate limiting won't scale.

- ***Performance Profiling in a Virtualized Environment***
  Jiaqing Du, EPFL, Switzerland; Nipun Sehrawat, IIT Guwahati, India; Willy Zwaenepoel, EPFL, Switzerland

Jiaqing Du said that clouds are built on lots of different hardware/software configurations, which provides significant opportunities for performance profiling/tuning in the virtualized environment. The problem is that, currently, profilers need to interact with underlying hardware quite a bit, and doing this is significantly more difficult in a virtualized environment. XenOProf is the only existing VM profiler, and it only works for Xen and requires admin access. Profiling in the guest can be done but doesn't tell one much about how the system is truly performing.

At a very high level most profilers work by keeping track of event counters and interrupts. The authors propose that these need to be exposed to guests through the standard PMU interface. Of course, one still needs to determine whether the VM should track that as CPU-switched—only in-guest execution is accounted to the guest—or domain switched, which includes other tenants on the same machine.

The authors implemented this method for KVM, using Intel VT extensions. Their experiment consisted of pushing packets to a Linux guest, running OProfile (in the guest), and monitoring instruction retirements.

How long will it take manufacturers to provide hardware support for virtualization? Du said they believe it will take a while, since there are many different hardware/software combinations. Thoughts about instrumentation-based profiling? These should work directly in the guest without the need for virtualization-specific techniques. How does this differ from XenOProf? Is there a more generic methodology? XenOProf does system-wide profiling and requires admin access. We've provided more options and don't require admin access.

- ***The Case for Energy-Oriented Partial Desktop Migration***
  Nilton Bila and Eyal de Lara, University of Toronto; Matti Hiltunen, Kaustubh Joshi, and H. Andrés Lagar-Cavilla, AT&T Labs Research; M. Satyanarayanan, Carnegie Mellon University

Currently PCs waste significant amounts of power while idling. Eyal de Lara said that sleep modes aren't used because users and IT expect always-on semantics, such as background apps and remote access. The authors aim to provide always-on semantics with the benefits of sleep mode. Potential approaches include migrating a VM to a consolidation server (like LiteGreen), but the problem is that VMs are very large and lots of memory will be needed on the consolidation server. Instead, the authors suggest just migrating the working set on that machine using a page-fault-based method.

When a machine goes to sleep a small VM is started on the consolidation server. Every time some memory access is attempted for memory not already present on the consolidation server, the server wakes the client, grabs those pages, and proceeds. The client falls back asleep shortly thereafter. The authors claim this allows for many images on the same server and relatively little network overhead.

However, in order for this to work well, sleep time needs to be long enough and the size of the memory footprint needs to be sufficiently small. The authors have conducted a feasibility study for their proposed methodology, using Snowflock. They have not actually implemented any of the migration code. They examined four workloads: login, email (every 10 minutes), IM client (messages every couple of minutes), and a multitasking workload (PDF, email, spreadsheet, file-browser, IM) and track the resulting page faults. Since they aren't prefetching (although an actual implementation would almost certainly do so), some of their results aren't always very good. However, their overnight test looks like things work nicely in low-usage settings.

The authors estimate a consolidation ratio of 9:1: that is, one can serve nine PCs on one PC, so roughly an order of magnitude more PCs could be consolidated onto a high-quality server. The remaining challenges include device methods that will avoid some of this power-cycling, including proactive methods, CAMs, and addressing policy questions as to when to migrate VMs back and forth.

What about disk accesses? De Lara replied that this requirement is an order of magnitude less than memory. Why do people need machines overnight? They don't, but IT likes these machines to be awake. We believe we leverage smaller sleep times. Have you considered partial wakeup modes? That would require massive stack changes. Wake-on-LAN is a very simple version of this.

- ***Energy Efficiency of Mobile Clients in Cloud Computing***
  Antti P. Miettinen and Jukka K. Nurminen, Nokia Research Center

Antti P. Miettinen pointed out that one of the major problems for mobile devices using cloud applications is battery life. One could do almost all of the processing locally (processor intensive) or do the processing remotely, essentially using the phone as a thin-client (radio intensive). The authors sought to understand the trade-off between computing and communication in terms of mobile phone power use.

Different cores have different energy profiles but, interestingly, mobiles actually use the least efficient cores. This is because mobile phone cores are selected for high peak performance on single-threaded applications. The straightforward solution is to leverage dynamic voltage and frequency scaling (DVFS). Essentially this means using cores that support lower power draw at decreased clock speeds.

With respect to radio power draw, the authors find that 3G radios are much more efficient at high bit rates, while WiFi is more forgiving of lower-bit-rate traffic. Consequently, particularly for 3G traffic, smooth traffic is handled with less power efficiency than bursty traffic. Moreover, if the core is set to a lower clock cycle (power saving on the CPU) while doing data transfer over the network, more power may be used overall, since the data rate of the 3G card will be slowed down accordingly (drawing more radio power).

As a very rough rule, the authors observed 1000 cycles core/one byte of radio parity. Of course, everything depends on the particulars. With a PDF viewer, the authors show power can be saved by running remotely on their hardware. There are lots of challenges: when to transition between thin and thick client operation on mobiles, tools for managing these transitions, energy-aware middleware, energy optimized protocols for thick clients, etc.

How did you do the PDF viewer offload? Miettinen replied that they ran evince on the server and exported x11, but this isn't an optimal thin client protocol, so they could do better (e.g., crawling the PDF is a big mess using x11). Someone else wondered if they had considered more complex apps (e.g., OCR, image processing, auto-translate). They hope that people will come up with these so they can study them. Current apps more or less prune out the computation.

ECONOMICS AND PRICING

Summarized by Alva L. Couch (couch@eecs.tufts.edu)

- ***CloudCmp: Shopping for a Cloud Made Easy***
  Ang Li and Xiaowei Yang, Duke University; Srikanth Kandula and Ming Zhang, Microsoft Research

Cloud computing involves difficult business choices. Cloud providers describe their services in incommensurate units, and it can be difficult to compare one service against another for a potential use. CloudCmp attempts to aid in this decision by combining service benchmarks with load profiles to create predictions of cloud performance without deployment. Speaker Ang Li concentrated on the first part: how to collect and utilize accurate benchmarks. This includes three kinds of measurement: storage and retrieval time, computation time, and network latency. Network latency was determined by measuring the time to query distributed data centers via PlanetLab. Studies of three unnamed cloud services indicated that different services have different performance strengths: one excelled at quick storage and retrieval, while another excelled at computation. The audience was concerned about why the three services were anonymous; Li responded that all three service agreements include prohibitions of reverse engineering, but that in one, disclosure of performance data was prohibited. Someone was also concerned that the measurements were made over short time periods and did not account for changes in background load.

- ***Distributed Systems Meet Economics: Pricing in the Cloud***
  Hongyi Wang, Microsoft Research Asia; Qingfeng Jing, Shanghai Jiao Tong University; Rishan Chen, Peking University; Bingsheng He, Zhengping Qian, and Lidong Zhou, Microsoft Research Asia

Pricing in the cloud is complex and—like pricing on first-generation timesharing servers—can vary with load. Pricing is fair if there is a balance between customer satisfaction and provider satisfaction. Hongyi Wang utilized 1/cost as a customer satisfaction index, and profit/cost (return on investment, or ROI) as a provider satisfaction index. By running instances of Postmark (storage-intensive), Parsec (compute-intensive), and Hadoop (communication-intensive) tasks on EC2, he depicted both fairness and variation in fairness due to load. He concludes that optimality points for customer and provider are often quite different: for Postmark, two VMs maximized customer satisfaction, while four VMs maximized provider satisfaction. Not surprisingly, satisfaction indices also varied with load, which he interprets as a kind of unfairness.

Someone questioned whether this kind of analysis will lead to fairer pricing from providers.

- ***See Spot Run: Using Spot Instances for MapReduce Workflows***
  Navraj Chohan, University of California, Santa Barbara; Claris Castillo, Mike Spreitzer, Malgorzata Steinder, and Asser Tantawi, IBM Watson Research; Chandra Krintz, University of California, Santa Barbara

A "spot instance" in EC2 is a server instance that is activated when the current (volatile) market price for service exceeds a predetermined (constant) customer-specified bid value. Spot instances are thus transient, are started when the market price lowers below the bid, and are terminated when the market price increases above the bid. Market prices are determined based upon supply and demand. Because of their transient nature, spot instances are cheaper

to use than "premium" instances that do not go online and offline based upon demand.

Navraj Chohan and his coauthors studied the use of spot instances during three kinds of Hadoop computations: word-count, sorting, and Monte-Carlo computation of π. Data for the computation was kept on ("premium") Hadoop HDFS nodes that were always running, while spot instances were allocated as extra workers. Even though spot instances are cheaper to use, using spot instances can be more expensive (and slower) than using only "Premium" instances, because of the time required to discover and correct the effects of terminated "spot" worker nodes.

Someone asked whether running spot instances with a high bid is equivalent to having a premium node, and whether market prices are periodic. Chohan responded that market prices show no predictable periodicity (and discussion groups on the Web report that spot instances can be volatile even if the bid price is always high enough, because Amazon can terminate them for reasons other than market price fluctuations, e.g., for maintenance).

- ***Disaster Recovery as a Cloud Service: Economic Benefits & Deployment Challenges***
  Timothy Wood and Emmanuel Cecchet, University of Massachusetts Amherst; K.K. Ramakrishnan, AT&T Labs—Research; Prashant Shenoy, University of Massachusetts Amherst; Jacobus van der Merwe, AT&T Labs—Research; Arun Venkataramani, University of Massachusetts Amherst

Disasters happen even to clouds. Tim Wood and his team considered how clouds should handle disastrous conditions. Business objectives for disaster handling include "recovery point objectives" (how much data can be lost), as well as "recovery time objectives" (how long it takes to resume processing of requests). Wood made some cost comparisons between cloud-based recovery and traditional co-located recovery. He estimated that recovery infrastructure for a RUBiS site with four servers and 99% uptime would cost $10,373 per year with co-located physical recovery servers and $1,562 if recovery servers are instead located on EC2.

As a second example, a data warehouse allows one to balance the cost of cloud recovery against recovery-point optimization, because one only pays for the servers when they are running to store recovery points. Open questions include how the provider can maximize profit (due to a similar mismatch between customer and provider satisfaction to that presented in another paper above), how many resources the customer should commit to disaster recovery, how to deal with correlated failures (e.g., regional outages), and how one should resume regular processing after a disaster is mitigated.

Someone asked whether the disaster recovery model included regional distribution of disaster recovery resources, and Wood responded that it did not. Would the mechanism continue to work if "everyone" did their disaster recovery this way? More resources would have to be available in the cloud for this to be practical.

- ***CiteSeer^x: A Cloud Perspective***
  Pradeep B. Teregowda, Bhuvan Urgaonkar, and C. Lee Giles, Pennsylvania State University

The Web site CiteSeer^x is supported by 22 physical servers, stores greater than 1.6 million documents and greater than 30 million citations, and responds to about 2 million hits a day. It includes a Web crawler, as well as software components for document conversion and ingestion, data storage, query response, and maintenance services. Pradeep Teregowda discussed the options for moving this site into the cloud, either in whole or in part. He considered two options, including Amazon EC2 and Google AppEngine. Moving an application to the cloud requires both data refactoring and code refactoring.

Data refactoring seems to be cost-effective, while the practicality of code refactoring remains unknown. In particular, the cost of moving code to AppEngine remains unknown. He concludes that for now, hosting static content and query response in the cloud is cost-effective, while the practicality of moving other software components is as yet unknown. Re-factoring the main software components may make cloud hosting of the entire application cost-effective.

Someone asked how CiteSeer^x will handle load changes, and Teregowda answered that, for now, the plan is for the cloud to handle only peak loads, while physical infrastructure will continue to handle non-peak loads.

NEW PROGRAMMING MODELS AND USAGE SCENARIOS

Summarized by Malte Schwarzkopf (malte.schwarzkopf@cl.cam.ac.uk)

- ***Spark: Cluster Computing with Working Sets***
  Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica, University of California, Berkeley

Spark is a framework for cluster computing optimized for iterative applications with fixed working sets, with datasets reused across multiple parallel operations. Conceptually, Spark builds upon frameworks such as MapReduce and Dryad, but improves their limited performance on iterative jobs such as those found in many machine learning and data-mining applications.

Spark is written in Scala and integrates with the Scala language for easy programmability. It defines the notion of a resilient distributed dataset (RDD), which can be obtained from the Hadoop Distributed File System or parallelized arrays. RDDs can be transformed using map and filter functions and are cached across operations. The main difference from previous models is that RDDs are defined to be persistent even across iterations within the driver program. Spark supports parallel operations of reduce, collect, and foreach

on RDDs and supplies shared variables (accumulators and broadcast variables) to allow for data to be communicated. For example, consider the use case of log mining: the logs would be RDDs, and there would be a driver node (running the main driver program) as well as a set of workers. The driver creates a distributed dataset, executes some filter operations on it, and then calls a cache() function. At this point, the RDDs exist as lazy objects on the cluster. On the first iteration of the algorithm, they are then disseminated to and cached on worker nodes. Further iterations will then hit the version cached in memory on the worker nodes, thus speeding up the information.

RDDs maintain fault-tolerant semantics as each RDD maintains lineage information that can be used to reconstruct lost partitions (RDDs). This is done by going back to the beginning and reapplying the filter and map operations (this restricts the programming model to disallow certain side effects, but provides fault tolerance).

In the evaluation, the authors compared Spark to the Hadoop open-source implementation of MapReduce and found that Spark outperforms Hadoop by a factor of up to 20 on the second and subsequent iterations, while running about 50% slower on the first iteration (Hadoop: ~127 seconds per iteration, Spark: 174 seconds for first iteration, 6 seconds for further iterations). There was no time for questions, due to a demo that interactively reproduced the above results.

- ***Turning Down the LAMP: Software Specialisation for the Cloud***
  Anil Madhavapeddy, University of Cambridge; Richard Mortier, University of Nottingham; Ripduman Sohan; University of Cambridge; Thomas Gazagnaire, Citrix Systems R&D; Steven Hand, University of Cambridge; Tim Deegan, Citrix Systems R&D; Derek McAuley, University of Nottingham; Jon Crowcroft, University of Cambridge

Anil Madhavapeddy described the implementation of a "fat-free" operating system that reduces the level of dynamic abstraction in computer systems. The authors point out that typical application stacks of modern operating systems consist of a large number of abstraction layers on top of one another, with dynamic languages adding another layer (language runtime) and virtualization another (hypervisor). Although undeniably convenient, these layers exist mainly for compatibility reasons and ensure legacy compliance. For example, as was pointed out, the POSIX layer was designed to run compiled and linked C applications, but not dynamic languages with garbage collection, such as Java or .NET code. Furthermore, the number of abstraction layers means that there is a greater potential for security vulnerabilities.

Madhavapeddy contended that the emergence of virtualization has produced a unique opportunity to remove as many compatibility abstractions as possible, as hypervisors provide a "stable" virtual hardware platform to develop against. This facilitates not only the removal of unnecessary abstractions, but also means that small research efforts that would previously have been doomed to fail due to the complexity of operating systems implementation can succeed. In addition to moving towards writing applications on top of the "bare metal" provided by virtualization architectures, the authors also propose to use the opportunity to introduce a strong and statically typed implementation language in order to avoid security vulnerabilities and improve code quality. Their language of choice is OCaml, which has strong static typing, is extensible, and has a simple runtime system introducing minimal overheads.

As a motivating benchmark, the authors reimplemented SSH and DNS in OCaml, and the measured performance is on par with the best, optimized existing implementations (OpenSSH and Bind, respectively). When turning on memoization in their DNS implementation, the authors even managed to reach an order of magnitude performance improvement over Bind. Coming from these motivating measurements, they have started implementing MirageOS, a new operating system that embraces the philosophy outlined above. MirageOS features zero-copy I/O and gets rid of multicore concurrency issues by only ever running one application. Application-level concurrency is provided by multiple VMs running on the same host system and communicating using message passing.

The authors evaluated an early alpha version of MirageOS in terms of SQL performance and memory usage and found that the performance is improved dramatically compared to a Linux-based setup, while the memory footprint remains the same. In further work, the authors are planning to investigate the use of MirageOS for highly specialized Web servers that can run at very high performance while using less resources than current setups. There was no time for questions.

- ***Scripting the Cloud with Skywriting***
  Derek G. Murray and Steven Hand, University of Cambridge Computer Laboratory

Derek Murray introduced Skywriting, a programming language for cloud computing. Existing frameworks such as MapReduce and Dryad have pioneered a managed approach in distributed computing, freeing programmers from having to concern themselves with the low-level details of message passing, synchronization, and fault tolerance. However, in doing so, they restrict the programming model to one that corresponds to a map-reduce paradigm (MapReduce) or a finite, static DAG (Dryad) and thus have difficulties modeling certain data flows. For example, unbounded iteration and recursive computation are impossible to express in either in any way other than submitting a series of jobs. Ideally, as the authors assert, one would like a truly universal programming model for cloud computing, allowing any Turing-complete program to be expressed.

With Skywriting, the authors have designed a programming language that can express any Turing-complete program. Its syntax is similar to JavaScript, but it is capable of express-

ing functional constructs such as lambdas. It is an interpreted coordination language and can delegate the actual computation to external high-performance code. Contrary to MapReduce and Dryad, it supports spawning tasks dynamically (using the spawn() primitive), as well as expressing data-dependencies through futures that can be dereferenced in the style of C pointers. In this way, Skywriting can express data-dependent control flow such as unbounded iteration to convergence.

The authors evaluated Skywriting by performing a micro-benchmark that tested the job creation overhead and found that it performs much better than Hadoop, which incurs up to 30 seconds of overhead (Skywriting: ~2 seconds). In order to evaluate Skywriting on real workloads, the authors implemented the Smith-Waterman string matching algorithm in Skywriting and ran it on a variety of cluster configurations. They found that the best performance for two strings of length 100,000 was achieved with 400 tasks on 20 workers with a speedup factor of about 2.5x compared to linear, but also that the system scales to cluster sizes of hundreds of nodes.

Anil Madhavapeddy asked whether Skywriting has an issue that needs fixing and how one would hook it into an existing cluster. For the former, Murray said that a possible weakness is that Skywriting does not natively support efficient data motion for MapReduce-type workflows (although it does support MapReduce in principle) and that it is a research prototype and much of the code is unoptimized (e.g., the task dispatch queue is single-threaded). For the latter question, Murray responded that Skywriting can easily be deployed onto a cluster using a set of scripts and that binding code to allow Hadoop or Dryad workers to be tied in with a Skywriting cluster is currently in development.

- ***Toward Risk Assessment as a Service in Cloud Environments***
  Burton S. Kaliski Jr. and Wayne Pauley, EMC Corporation

Formal risk assessment is a necessity for many commercial outfits nowadays, but tends to be a highly time-consuming service. There are a number of well-established standards, but all assessment is still done manually by humans. Wayne Pauley asserted that as a consequence of this, traditional risk management strategies fail when applied to the cloud, which is a fast-paced, dynamic environment that is also geographically diverse and on-demand. Endpoint devices can be anything and resource pooling means that it is impossible to tell in advance what resources are going to be shared with. Subcontracting at the cloud provider and the challenge of having to meter and monitor customers while avoiding leaking private data through observation of usage characteristics are further issues.

The authors introduced the notion of "risk assessment as a service," in a manner akin to the automated credit ratings in use today. Various different models are possible to imagine: self-assessment by the cloud provider, third-party audit, or consumer assessment involving internal and external agents. To facilitate this, the authors have designed a risk assessment architecture for the cloud, with a risk monitor and a set of agents as its core components. The risk monitor is supplied with information from a variety of agents (both with the cloud and with customers) and also receives information from external auditors and definition lists.

After having sketched the architecture, Pauley now sees their future work in figuring out what sensors are needed with the agents, what implementation language to use for the system, and working out how customers and providers can react to the assessment in an automated fashion. Furthermore, they plan to undertake an evaluation of the effectiveness of automated assessment versus traditional (manual) methods, as well as of the actual trust assurances given by the automated measurements.

How much money should be spent on security assurances and can risk be measured in a monetary form? Pauley replied that some equilibrium needs to be found; currently, there is not that much intellectual property in the cloud, but as we start moving more information into it, the importance of determining the value of individual assets in order to assign risk thresholds and monetary value to it will grow.

- ***Information-Acquisition-as-a-Service for Cyber-Physical Cloud Computing***
  Silviu S. Craciunas, Andreas Haas, Christoph M. Kirsch, Hannes Payer, Harald Röck, Andreas Rottmann, Ana Sokolova, and Rainer Trummer, University of Salzburg, Austria; Joshua Love and Raja Sengupta, University of California, Berkeley

Christoph Kirsch described using virtualized flying vehicles carrying sensors as information acquisition nodes that can be sold using a similar elasticity model to other resources in cloud computing. The authors are prototyping this concept using an autonomous quadrocopter, called JAviator, as their hardware platform. The quadrocopter is controlled by a computer and must be controlled this way, as the platform is unstable without computerized control. The quadrocopter acts as a "cyber-physical server," having an IP address and a geographic location, plus the capability to move about and carry sensors. Multiple quadrocopters can form a "cloud." Kirsch went on to introduce the notion of a "virtual vehicle" that can migrate between physical vehicles (which can run multiple virtual vehicles at the same time). One use case for this is as follows: imagine there are a number of flying vehicles that follow predefined routes. If someone now were to require a vehicle that describes a flight route that none of the physical vehicles describes individually but which a combination of them could cover, transparently migrating a virtual vehicle between different physical vehicles enables us to provide this.

In the experimental setting, real vehicles with real sensors (Webcam, laser, ultrasonic, gyroscope, accelerometer, etc.) exist, as well as real servers that are mounted onto real vehicles. The latter are still works-in-progress and will provide

a powerful small form-factor server more powerful than an embedded system. Virtual vehicles can run on those physical vehicles and servers and have access to virtual sensors (also still a work-in-progress—current efforts are focused mainly on the Webcam) as well as virtual processors (which provide real-time guarantees required for flight control). Migration of virtual vehicles between different physical vehicles needs to be really quite fast (~10ms), due to the short reconnaissance times the physical vehicles experience. Furthermore, there is the notion of "virtual actors," which substantiate themselves in the form of virtual vehicles and which can pilot real or other virtual vehicles.

The current research effort is split between Salzburg (virtualization infrastructure) and Berkeley (collaborative control), with joint work on the programming language. In the virtualization infrastructure, an "Earliest Deadline First" (EDF) scheduler was added to Xen to support temporal in addition to spatial isolation, and future work is concerned with power isolation, migration, and tracking real and virtual vehicles. The collaborative control problem is mainly concerned with the allocation of real vehicles to virtual vehicles under consideration of mutable flight plans (read-only flight plans for physical vehicles are easy, but read-write flight plans for virtual ones are hard due to potentially conflicting interests of virtual vehicles). The programming language in use is the Collaborative Sensing Language (CSL), which will specify dynamically changing missions of virtual vehicles. The key challenge here is said to be the handling of concurrent and dynamically changing sets of real and virtual vehicles.

How flexible are virtual vehicles and are there any other cyber-physical systems that the authors know about? Virtual vehicles are quite flexible, but of course ultimately must conform to the limits of real vehicles that can only fly in certain ways (in addition to other challenges). To the second question, "anything that moves" is potentially a cyber-physical system; however, the applications considered in this case require certain computational power, which excludes some options (e.g., very lightweight sensor nodes).

SECURITY AND RELIABILITY

Summarized by Rik Farrow (rik@usenix.org)

- ***A First Look at Problems in the Cloud***
  Theophilus Benson, University of Wisconsin—Madison; Sambit Sahu, IBM Research; Aditya Akella, University of Wisconsin—Madison; Anees Shaikh, IBM Research

Benson explained that they studied three years of support data of an Infrastructure as a Service (IaaS) cloud provider. The data was from a support forum where a new thread was treated as a trouble ticket, and resolution could come from other users or the IaaS support group. They used an automated information retrieval algorithm (Lemur) for extraction of problem clusters, then selected a subset of these clusters for manual analysis.

The authors found that most problems with the cloud service fell into five categories: image maintenance, connectivity, performance, virtual infrastructure, and application-related. Over the three-year period, image maintenance problems declined as better APIs and tools appeared for dealing with images. Problems with virtual infrastructure increased whenever new features, such as cloud storage, appeared. Over time, fewer operator interventions occurred as users became better able to solve their own problems as the online support database grew in size.

Benson gave an example of a problem that required the cloud support to intervene. A user complained of losing connectivity with her instance, and it turned out the VM instance was running out of memory and killing the ssh daemon. Benson suggested that cloud providers expose more information to their users without divulging infrastructure details while avoiding storage overhead by collecting more data/logs. He also suggested adding more user controls.

John Arrasjid suggested that they should have test plans from the provider's developers ready when new releases come out. Benson agreed, saying that would be a good way to decide which tools to expose to users. Christina Serbin asked about a spike in a graph that occurred in March 2009. Benson replied that this was an anomaly.

- ***Secure Cloud Computing with a Virtualized Network Infrastructure***
  Fang Hao, T.V. Lakshman, Sarit Mukherjee, and Haoyu Song, Bell Labs, Alcatel-Lucent

Fang Hao laid out their goals: isolation transparency (see only the user's own virtual network), location independence (locate anywhere in the data center), easy policy control (change policy settings for cloud resources on the fly), scalability (restricted by total resources available), and low cost (use off-the-shelf whenever possible). VLANs have been used to provide network isolation, but this solution has problems. The VLAN ID field only supports 4096 VLANs, and hypervisors must be configured to map VLANs to particular VMs, a potential weakness, as hypervisors can be attacked.

Their solution involves adding Forwarding Elements and a Central Controller. FEs are Layer 2 routers that enforce forwarding of packets to a particular edge network and interface. FEs attach edge networks to the core networks and route packets between core networks. Users decide which VMs can communicate, and the Central Controller configures the FEs.

Hao described an attack that used traceroute to determine domain 0 addresses and looked for other numerically close addresses with a short roundtrip time. Their solution prevents this attack, because only addresses that are part of a virtual network can be seen, since the FEs control the forwarding of packets.

Someone asked about the delay imposed when a host first issued an ARP for an address, an operation that the Central

Controller must handle. Hao responded that this occurs in the data center, so latency will be low. Also, this only occurs once, for the first packet. The same person asked about how they came up with the Layer 2 mechanisms, and Hao answered that it is basically a hash reuse process, and that they really didn't evaluate performance. Wayne Pauley asked if removing traceroute increased the number of support requests, and Hao answered that traceroute is still available, but can only be used to view the customer's own virtualized network.

- ***Look Who's Talking: Discovering Dependencies between Virtual Machines Using CPU Utilization***
  Renuka Apte, Liting Hu, Karsten Schwan, and Arpan Ghosh, Georgia Institute of Technology

Renuka Apte described a system for uncovering relationships between VMs by examining CPU usage. Knowing which VMs have dependencies is useful when it comes to migrating VMs, as you don't want to move related VMs too far apart, where "far" has to do with network latency.

They use xentop to monitor CPU utilization, at a frequency of once a second with a window of 300 seconds. Both of these values can be adjusted, but this is what they found worked well in their experiment. Then they used k-means for clustering spikes in CPU utilization. The value of k must be supplied by the user but should match the number of applications sharing dependencies. They found they could identify dependencies with 91% true positives and 99% true negatives. They ran three applications in their test, with multiple instances of RUBIS and one of Hadoop, with one master and three slaves.

Someone asked what the meaning of false positives was, and Apte answered that it meant identifying a particular dependency that didn't exist. This would be harder to measure in the real world, where dependencies are not known in advance. Someone else suggested looking at network traffic instead of CPU load, and Apte pointed out that VMs can share the same physical system and not have any external network traffic. You would also need to clean up traffic traces to remove any non-significant traffic when resolving dependencies.

- ***A Collaborative Monitoring Mechanism for Making a Multitenant Platform Accountable***
  Chen Wang, CSIRO ICT Center, Australia; Ying Zhou, The University of Sydney, Australia

The speaker for this presentation was held up because of visa issues, so the session chair made a brief summary of the paper. The authors use Merkel B-tree, which is authenticated so you can present evidence back to the cloud provider. The goal is to provide clients of a multitenant service, such as force.com, with a means of providing evidence that SLAs have not been met, for example.

PANEL

Summarized by Alva L. Couch (couch@eecs.tufts.edu)

- ***Barriers to Cloud Adoption and Research Opportunities***
  Moderator: Erich Nahum, IBM T.J. Watson Research Center

  Panelists: Albert Greenberg, Microsoft Research; Trent Jaeger, Pennsylvania State University; Orran Krieger, VMware; Prashant Shenoy, University of Massachusetts Amherst; Ion Stoica, University of California, Berkeley

Trent Jaeger discussed the misconceptions common to cloud security. The cloud provider thinks of security as "guards" around the user's data, which is processed within the cloud, while a security expert thinks of data as something that should be encrypted whenever it is stored in the cloud. Neither of these is practical. We need an intermediate approach in which data is secure in the cloud but computation can still occur inside the cloud. The key to this approach is some form of transparent security where data remains both secure in the cloud and available to applications. This requires a number of security measures, including proofs of host "correctness."

Orran Krieger discussed the role of clouds in "fungible" computing. A "fungible" asset is a commodity whose provider can be freely changed without impact. If clouds become fungible, then cloud providers can compete with host applications without incurring refactoring costs. Fungible computing is a transformational paradigm. A new company or startup can use the cloud as a cost-effective way of experimenting without capital investment, in the sense that no capital equipment is either purchased or managed to create, for example, a new Web site. When a company fails, its images are deleted, so there is little overhead incurred for failure.

There are two competing paradigms for clouds: vertically integrated clouds like Amazon, AppEngine, Azure, and IBM, and the cloud "marketplace" of a plethora of VM hosting services. Krieger hopes that the "marketplace" wins. For this to happen, we need common abstractions for writing virtualized applications, as well as practical methods for federating services to be used by the applications.

Krieger envisions future transformational features, including a "follow-me-anywhere" desktop, laptops with remotely administered system administration and security features, and even the ability to be "a sysadmin for your mom." No one has even started tackling the "tough problems."

Prashant Shenoy discussed three challenges of cloud computing, including economics, manageability, and network/cloud interoperability. The cloud argument is that leasing is cheaper than owning. But in fact, this is built into outsourcing agreements and one who outsources IT does not interact with cloud economics directly. Those who do must deal with new challenges, including resource provisioning. There

remains a need for an economically justifiable private cloud model.

Second, there is a need for enterprise management tools such as IBM Tivoli and HP OpenView to understand and be able to manage the cloud, as one way to make private clouds economically justifiable for large enterprises.

Third, there is a need for coordination between cloud management and the network in which the cloud functions. Optimizing cloud application configuration requires also adjusting network configuration, including available bandwidth. Enterprise management tools such as IBM Tivoli and HP OpenView must be extended to manage the cloud and should also be able to tune the cloud, applications, and network as one integrated task.

Ion Stoica discussed the need for meaningful service level agreements (SLAs) in selling and consuming cloud services. SLAs provide the customer with "one throat to choke" when things go wrong, and they serve as a contract and point of accountability between customer and provider. In this context, research opportunities include achieving high utilization (for the provider) in the presence of interactive applications, providing appropriate isolation between applications, and the ability to scale up and down. The "holy grail" is that an application can safely assume that it is running in isolation.

A second issue is "multi-datacenter support" for cloud applications. To achieve appropriate availability and scalability, we need a shared API for applications that can be provided at several places to support migration. Research challenges include intelligently choosing locations for an application, designing an appropriate API for interactions with the data center, and assuring data consistency, no matter what happens.

Finally, the cloud needs to provide appropriate and usable notions of data security that allow applications to safely execute across administrative domains. Research opportunities include how to construct privacy-preserving queries, how to utilize encrypted data, and how to leverage test-case management (TCM) in the cloud environment.

Moderator Erich Nahum then pointed out that the panel was way too much in agreement. In response, Krieger claimed that the whole security paranoia about clouds is overblown. In fact, the same security problems affect non-cloud systems and customers don't seem to care. Trent Jaeger pointed out (in agreement) that in the 1990s we had agent computing and sent code "into the wild" to be executed. We again have a mental hurdle to leap, in writing code to be executed on systems one does not own.

Sambit Sahu asked about trusted security structures. What can one do to analyze security when one cannot examine the software stack in the cloud? Jaeger asked how one is even going to know what is running. Krieger responded that you can know what you are doing, but not what is running beside you.

Someone then asked what utility computing regulations should be. Krieger responded that providers must be regulated, but some of the things that turn utilities into monopolies will not happen in this case. There will be a market for third-party auditors to ensure compliance. Shenoy pointed out that for clouds, the main issue is not going to be regulation but, rather, compliance with standards, driven by customer needs.

Someone asked about the new kinds of security attacks that arise from "inside" the cloud. Krieger predicted that the larger enterprises are not going to have this problem, because they are going to be running their own private clouds. The smaller enterprises, which cannot justify private clouds, are going to be the ones at risk. The need is to provide security to smaller enterprises that—in fact—they would not be able to afford to provide for themselves "in a million years."

Someone responded that the inside attacker has access. Krieger responded that in the future, the attacker won't have access, and that there are technical solutions to this problem. Krieger pointed out that the cloud is awesome for Web apps that need to scale, but—as a whole—looks more like a 24/7 enterprise application with nearly constant load and potentially low (10%) utilization, due to the need to respond to demand changes. We would like to think of it as an 80% utilization, but this may not be realistic.

Someone next asked how one migrates to the cloud. Shenoy responded that the key is to understand your own cloud usage strategy. Krieger responded that he is not sure whether there is a problem for larger enterprises, because enterprise data centers are already virtualized and anything that will run there will run in the cloud. The appropriate migration strategy is to make the cloud look like the enterprise from which the application was migrated. Shenoy pointed out that virtualization is already ubiquitous outside the cloud. Stoica said that getting out of the cloud is straightforward; one just duplicates the API outside, and only when it becomes cost-effective.

A member of the audience admitted to some remaining confusion about how to define the cloud. Krieger reminded us of the story of the blind men and the elephant. It is not that interesting to define the term "cloud" as any more than we have done to support the Web so far. Even the very simple model of IaaS—if it can be standardized—is going to be transformative.