usenix ;login:
OCTOBER 2012 VOL. 37, NO. 5
ELECTRONIC SUPPLEMENT

Conference Reports

4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '12)

Boston, MA
June 12-13, 2012

Opening Remarks

Summarized by Rik Farrow (rik@usenix.org)

Dave Maltz started the workshop by explaining the format for presentations. Only questions specific to the presentation should be asked at the end of a talk. When a session's talks are completed, all speakers return to the podium to answer questions.

In these summaries, some scribes incorporated both specific and discussion questions with the paper summaries, while others included a specific discussion section.

Cloud Risks

Summarized by Rik Farrow (rik@usenix.org)

The Seven Deadly Sins of Cloud Computing Research

Malte Schwarzkopf, University of Cambridge Computer Laboratory; Derek G. Murray, Microsoft Research Silicon Valley; Steven Hand, University of Cambridge Computer Laboratory

Malte Schwarzkopf gave a lively presentation about current failings in cloud research that come about largely due to shortcuts and simplifications. Cloud computing is large-scale parallel processing, what we used to call cluster computing and scalable Web app serving, and has become the rage. Schwarzkopf said that their "seven deadly sins" are, for the most part, easy to fix.

Schwarzkopf only presented the first three sins due to time constraints. He described the first sin as unnecessary distributed parallelism. People use parallelism to solve performance problems, but parallelism isn't free since it requires coordination and has other overheads. In their survey of current research, the authors found diminishing returns as scale increases and because of the overheads required in distributed parallelism. In some cases, a single system can do better than a distributed one, especially when the data set is small enough to fit entirely in memory of one of today's servers.

Schwarzkopf described the second sin as imagining that the cloud is homogeneous. In reality, clusters used for testing may have a changing background of batch jobs running concurrently. If the test cluster is composed of EC2 instances, there will be many influences under neither the control nor the visibility of the researchers. For these reasons, each run needs to be repeated five to ten times, and the results presented with error bars.

The third sin is comparing your research results against Hadoop. Hadoop is general by design, and by designing software to do one task well you will certainly do better. Schwarzkopf used a comparison of CIEL (a Cambridge project) against Hadoop as an example. CIEL does better than Hadoop, and when using arrays instead of data streams, CIEL array performs orders of magnitude better than Hadoop. But MPI is faster still as well as scaling better than CIEL array.

Terence Kelly (HP Labs) quoted Rob Pike as saying, "Systems research is irrelevant." He pointed out that Schwarzkopf had focused on performance. Schwarzkopf replied that you do get performance by investing more work, but this tradeoff is poorly analyzed. We often assume that distributed parallelism is the way to go, even while it is much easier than MPI programming.

Icebergs in the Clouds: The Other Risks of Cloud Computing

Bryan Ford, Yale University

Bryan Ford also rained on cloud computing in general, but asked that listeners come away with a constructive interpretation of this gloomy talk. Bryan pointed out that there are some well-known risks of cloud computing: security of data, integrity of data, privacy, malware defense, availability, and reliability. But Bryan listed five more issues.

First, several researchers, including himself, had demonstrated information leakage via side-channels. The second issue is reactive stability, and Bryan provided an example. Suppose someone's cloud application relies on another cloud provider for a load balancer, and the entire application runs within an infrastructure provider. The infrastructure provider has its own secret sauce in the form of power optimization, which is hidden from the application provider. These optimizations are not stable and convergent and can create positive feedback loops. At this point, Bryan displayed an image of the Seattle Narrows Bridge, which was famously unstable in moderate winds to the point of collapse.

Bryan described the third issue as cross-layer robustness when people typically build in layers. Suppose the application provider wants reliability to be five nines (99.999%), but uses two cloud storage providers who guarantee only three nines. What the application provider doesn't know is that both cloud storage providers rely on the same network provider, so one network failure could cut out both storage providers. Bryan's fourth issue is that we are assuming that we will always be connected. He compared the state of affairs to the hive mind

in *Ender's Game,* where disabling the hive mind causes the collapse of an empire.

Bryan ended by wondering if we ourselves are the bad guys. Will distributed apps still exist in any form in 1000 years? Book, music, and video produces have so far provided the option of a copy that can be preserved. But cloud-based artifacts destroy this archive property. And newer artifacts, such as cloud-based games like World of Warcraft (WoW) or search engines, have become part of our culture, but are not archivable.

Dave Maltz (Microsoft), the session chair, started the discussion, by asking the two presenters if they had more points to make. Schwarzkopf responded to Bryan's final issue that the German government compels the owner of any blog that reaches a level of importance to provide an archive of the blog to the government. Bryan said that this sounded like the right direction, but a blog is passive compared to WoW. John Sopka (EMC) said tracking every artifact in WoW would require massive storage, and didn't think it has ever been done or needed to be done. Bryan responded that it was a good research opportunity, even if all that was preserved was a single snapshot each year.

Byung-Gon Chun (Yahoo! Research) wondered if when Schwarzkopf said Hadoop, he really meant the MapReduce layer on Hadoop. As a community, we need to learn how to provide relevant comparisons. Schwarzkopf replied that he had conflated Hadoop and MapReduce in his talk, but this combination is widely used in research. Schwarzkopf said that Hadoop can be optimized big time, and that we are obsessed with performance. Dave Maltz followed up by asking whether it is all about performance? That's what lots of people are optimizing for. Schwarzkopf replied that if there was something else as easy to use as Hadoop and MapReduce, people would be using it. He would prefer to be using a different programming model himself.

Dave Maltz asked about the assumption of heterogeneity in the cloud, and Bryan responded by pointing to a nice, small body of research for dealing with persistent tail inequality. What would be wonderful would be to come up with a general scheme for dealing with heterogeneity issues. Schwarzkopf emphasized Bryan's point by mentioning a recent Google published cluster trace that showed 12 different types of machines in one cluster. If you are running a big cluster and are doing rolling upgrades to replace machines, you will definitely have different types of machines in your cluster. John Sopka said that heterogeneity has been a problem for years, and the aware consumer should demand some level of quality of service. Most issues Bryan raised assume that the provider will be fair, but we need educated consumers. Schwarzkopf responded that as a researcher, you have to ensure that your experiments can be repeated with similar results.

Cloud Hardware

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

Saving Cash by Using Less Cache

Timothy Zhu, Anshul Gandhi, and Mor Harchol-Balter, Carnegie Mellon University; Michael A. Kozuch, Intel Labs

Timothy Zhu presented work which looks at the unconventional idea of scaling down the caching tier for multi-tier cloud applications, during periods of low loads. The main motivation for this work is that DRAM is quite expensive, so scaling down the caching tier can significantly reduce operating costs. In fact, Timmy mentioned that for a typical 4:1 ratio of peak load to low load, the authors were able to shrink the caching tier by 50%.

Prior work has looked at scaling down the stateless application tier during low loads, but there is no prior work looking at scaling down the stateful caching tier, possibly because of the seemingly small savings this would afford. Timmy's results show that the reduction in required hit-rate at the caching tier during low loads is usually small (10%), but the subsequent cache size reduction is significant (50%), because of the skewed nature of the (Zipf) popularity distribution of Web items. Of course, on the practical side, one needs a mechanism to shrink the caches without losing the hot data on those caches. Timmy suggests that one simple way of shrinking the caches is to move the hot data off of the retiring caches by treating them as a second-level cache temporarily, before taking them offline.

Tim Wood from George Washington University asked how long it takes to scale down the cache, given that data needs to be moved off of the retiring cache. Timmy replied that the answer depends on the system under consideration, and for his small-scale system implementation, the caches could be scaled down in a matter of minutes. Timmy also mentioned that their HotCloud paper includes a theoretical model that estimates the time required to move the data off of the retiring cache. Tim Wood then asked whether the work assumed turning off cache instances or simply resizing the existing cache instances, and whether Amazon EC2 allows for resizing of instances. Timmy replied that their work assumed turning off instances, and he believes that Amazon does not allow resizing instances on the fly. Weisong Shi from Wayne State University expressed his concern on the degradation of performance when scaling down the caching tier. Timmy clarified that the work takes performance into account by looking at a mean response time target that the system has to meet at all times. Since the caching tier, which is typically provisioned for peak load, is scaled down when the load is low, performance is not compromised. Ryan Mack from Face-

book asked whether they looked at consistency issues or not. Timmy replied that they focused on a read-only workload, and so consistency wasn't a concern. However, Timmy mentioned that the authors are considering consistency as part of future work, which will look at write workloads as well.

Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2

Zhonghong Ou, Hao Zhuang, Jukka K. Nurminen, and Antti Ylä-Jääski, Aalto University, Finland; Pan Hui, Deutsch Telekom Laboratories, Germany

Zhonghong Ou presented this work which looks at hardware heterogeneity in the Amazon EC2 platform. Zhonghong and his co-authors performed a bunch of experiments across various instance types (small, large, extra-large) on Amazon EC2 over a period of a few months, and observed significant hardware heterogeneity within the same instance type. Further, hardware heterogeneity was also present across availability zones, which are basically distinct and insulated locations that protect applications from a single point of failure. Results show that performance can vary by as much as 60% because of hardware heterogeneity for CPU-bound (UnixBench), memory-bound (Redis), and I/O-bound (Dbench) workloads.

In order to exploit this heterogeneity, Zhonghong proposed a simple (and controversial) trial-and-error idea that aims to reduce EC2 rental costs. The idea is to try out an EC2 instance, and if it is not performing well, then drop that instance and apply for a new one. Even if you only restart an instance twice, this trial-and-error idea can reduce rental costs by as much as 40%. Of course, the cost savings depend on the running time of the job. Further details on this dependence can be found in the paper.

The presentation attracted a lot of questions from the crowd, and a few of the interesting ones follow. Bryan Ford from Yale asked how much the performance variation differs from processor to processor. Zhonghong answered that the variation for the 5645 processor was much higher than that for 5507 and 5643. Bryan then asked about the impact of heterogeneity. Zhonghong replied that there is a lot of heterogeneity in cloud instances, and, in fact, many Netflix users (Netflix uses Amazon EC2) are annoyed by the variation in performance they get. Dilma Da Silva from IBM Research added that IBM faced a similar complaint from some of their users, and so IBM added some notion of lower and upper bounds on performance. Masoud Moshref Javadi from USC asked about heterogeneity in network performance. Zhonghong mentioned that they had looked into this but they found that variation in network performance depends on a lot of factors, such as location and network topology, and thus is quite tricky to analyze. Masoud commented that cloud service providers should also guarantee some lower bound on network performance.

Rodrigo Fonseca from Brown University wondered what would happen if all customers started using a trial-and-error sort of idea for getting instances. This question obviously drew a lot of laughter from the crowd. Zhonghong admitted that they got this question from the reviewers as well. He mentioned that they hadn't looked into this yet, but this would be a starting point for their future work. A follow-up comment was that service providers must provide homogeneous performance to all customers in order to discourage such tricks. Bryan Ford made a witty comment that service providers could simply hide the CPU ID information from customers so that they don't find out about the heterogeneity in the first place.

RAMCube: Exploiting Network Proximity for RAM-Based Key-Value Store

Yiming Zhang, National University of Defense Technology; Chuanxiong Guo, Microsoft Research Asia; Rui Chu, National University of Defense Technology; Guohan Lu, Yongqiang Xiong, and Haitao Wu, Microsoft Research Asia

Yiming Zhang presented this work which basically looks to extend RAMCloud (John Ousterhout et al. from Stanford University). RAMCloud is a RAM-based key-value store which is designed to be persistent by using a combination of backup nodes and recovery nodes. However, RAMCloud was not tailored for datacenter networks. RAMCube looks to enrich the RAMCloud approach by exploiting the design of datacenter networks. The main goals of RAMCube are fast failure recovery (1–2 seconds) and high performance.

The key idea leveraged by the authors is that datacenter network topology is usually known, and need not be treated as a black box. Thus, the recovery nodes assigned to a primary node are 1-hop neighbors. Likewise, the backup nodes are 1-hop neighbors of the recovery nodes. The calculations of the number of recovery nodes and backup nodes can be found in the paper. For failure detection, RAMCube uses heartbeat messages and for failure recovery of primary nodes, the recovery nodes are used in conjunction with the backup nodes. Because of the 1-hop proximity of nodes, RAMCube can recover 64 GB of data in 1–2 seconds, assuming 10 Gbps network bandwidth and 100 MBps disk bandwidth. A prototype of RAMCube was evaluated on a 16-server testbed running Windows 2008. The experiments demonstrated quick failure recovery, graceful performance degradation and almost linear scaling of throughput with the number of clients.

Raja Sambasivan from Carnegie Mellon University asked how long it took for failure detection. Yiming answered that failure detection required only a few milliseconds

since RAMCube has a 1-hop proximity for its nodes, unlike InfiniBand-based systems. Bryan Ford from Yale asked how RAMCube deals with correlated failures. Yiming replied that RAMCube has recovery nodes on different racks, so that provides some protection from correlated failures. Ryan Mack from Facebook commented that sometimes an entire datacenter can go down as well. Someone asked how RAMCube is different from DHT-based solutions. Yiming replied that the main difference was the 1-hop proximity of the primary and recovery nodes and the recovery and backup nodes.

Networking

Summarized by Theophilus Benson (tbenson@cs.wisc.edu)

Opening Up Black Box Networks with CloudTalk

Costin Raiciu, Mihail Ionescu, and Dragos Niculescu, University Politehnica of Bucharest

Costin Raiciu showed how cloud customers can use active probing tools, such as traceroute, to discover a cloud provider's network topology. Costin then described the discovered Amazon EC2's network topology as a VL2-style multi-rooted tree with full-bisection bandwidth. Using this topology information, he optimized the performance of several of his applications. Based on his observations, Costin proposed CloudTalk, a system that allows the cloud provider to share topology information with the cloud costumer without the need for active probing.

CloudTalk allows a tenant to represent an application as a transfer or a set of flows with characteristics ranging from bandwidth requirements to application end points. Given this information, CloudTalk simulates the network and calculates an approximate completion time for the transfer or flows. Customers can use CloudTalk to improve their application's performance by querying the cloud provider for the completion time of the application given different placement decisions and by choosing the placement decision with the lowest completion time.

An audience member asked if CloudTalk can be constantly run in the background with the results from a prior run used to answer queries from low-latency applications such as Web applications. Costin explained that he implemented a similar idea during his Web-application experiments; CloudTalk was used only for initial placement of the Web. Bryan Ford from Yale wanted to know if Costin had explored other approaches in the Internet mapping world, such as iPlane, and how customers could collaborate to create such a mapping. Costin explained that this would require a significant number of customers to participate for such a mapping to provide relevant value.

GRIN: Utilizing the Empty Half of Full Bisection Networks

Alexandru Agache and Costin Raiciu, University Politehnica of Bucharest

Alexandru Agache described how full bisection networks are perfect when most servers send data to each other; however, in most real networks only a few servers are sending data. Thus the network is extremely idle and yet, in spite of this fact, many servers are unable to efficiently share network resources. Alexandru proposed developing a scheme to improve the network utilization by using only MPTCP and additional server ports. He designed GRIN, an architecture whereby each additional server port is used to connect the server to other servers within the same rack. In GRIN, a server is able to better utilize the network by sending traffic through an intermediate server. The links used to directly connect servers to each other are called GRIN links.

Alexandru simulated GRIN on a VL2 topology with 120 servers and compared GRIN with a multihomed topology. The experiments showed that under a random permutation matrix, GRIN's performance degraded faster than the multihomed topology because each server is more likely to find its GRIN links preoccupied as the number of servers utilizing the network increases. However, with the multihomed topology, each server is guaranteed exclusive access to additional uplinks. With an incast traffic matrix, GRIN performance degraded slower than the multihomed topology because the GRIN links provided significantly more capacity than the multihomed topology. Alexandru concluded by describing how GRIN is a simple and cheap proposal for increasing utilization of the network bandwidth.

An audience member wanted to know how bandwidth utilization would be affected by creating GRIN links between servers in different rack switches. Alexandru explained that connecting servers in different rack switches would result in cabling issues and would require complex routing algorithms.

EyeQ: Practical Network Performance Isolation for the Multi-Tenant Cloud

Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, and Balaji Prabhakar, Stanford University; Changhoon Kim, Windows Azure

Vimalkumar Jeyakumar described how the performance issues within public clouds forced Netflix to re-architect its applications once they adopted a public cloud. Despite problems like this, cloud providers do not admit to having performance issues and, even worse, they don't provide customers with performance guarantees. Vimalkumar explained that in the ideal scenario, a cloud provider would provide each VM with a minimum bandwidth requirement. However, this is currently not achievable since existing network QoS primitives do not scale to cloud requirements, and it is hard to give requirements at short time scales.

Using an experiment, Vimalkumar demonstrated how a spiky UDP flow impacts a steady TCP flow. The problem occurs because the UDP flow fully utilizes the link and forces the TCP flow to back up. Motivated by this, Vimalkumar developed EyeQ. The key idea behind EyeQ is to prevent full link utilization by creating artificial bandwidth headroom (bandwidth throttling) at the receiving server. Artificially throttling a link at the server prevents the network link from being fully utilized and allows congestion to be detected before it hits the network. EyeQ consists of two components: a rate limiter at the sender and a congestion detector at the receiver which creates the bandwidth head room, detects congestion, and sets the sender-side rate limiters based on the amount of congestion detected.

An audience member asked if Vimalkumar would once again explain the bandwidth headroom? Vimalkumar responded that EyeQ currently leaves a 10% bandwidth headroom on server uplinks, thus restricting server links to 90% utilization. Vimalkumar further described that EyeQ measured link utilization at a fine time scale, and when utilization reached 90%, EyeQ would send information to the sending server's rate limiter to limit the sender's sending rate. Another member asked about the processing overhead of fine-grained monitoring. Vimalkumar responded that these calculations are simple, require little state, and can be performed on hardware in network interface cards using a simple counter.

A Case for Performance-Centric Network Allocation

Gautam Kumar, Mosharaf Chowdhury, Sylvia Ratnasamy, and Ion Stoica, University of California, Berkeley

Gautam Kumar explained that highly parallel applications are written in high-level languages oblivious of the network support required for the jobs. Gautam then explained how these jobs have only a few network communication patterns, and thus it is feasible to expand the language's framework to make decisions about how to determine the support required for the network's communication patterns. Current approaches to expanding the frameworks either focus explicitly on fairness or performance. Gautam illustrated through examples how different network communications patterns achieve significantly different performance under different network-sharing approaches as they scale up. Thus different solutions are required for different network communication patterns to ensure better sharing.

Gautam illustrated that the appropriate allocation strategy for broadcast is flow allocation, while for the shuffle pattern, proportional allocation is the appropriate solution. In experiments with jobs containing different communication patterns, Gautam showed that his framework is able to allocate the appropriate scheme for each type of communication pattern and allow jobs to finish within comparable time periods ever when scaled up.

An audience member asked what sort of jobs he examined and why shuffle took a large portion of the time. Gautam responded that he didn't know the exact type of jobs, but the traces that he examined were from Facebook traces. Another audience member asked whether Gautam's framework supported jobs with different priorities and how he would extend his framework to support priorities. Gautam reframed the question in terms of the number of resources given to each job and described how the underlying formulation for his framework can be changed to introduce a scaling factor that would support priorities.

Programming Models

Summarized by Rik Farrow (rik@usenix.org)

Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters

Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica, University of California, Berkeley

Matei Zaharia explained that working with large data streams has usually meant spending a lot of resources for resiliency in case of node failure. Examples of data streams include user activity statistics, spam detection, traffic estimation, and network intrusion detection. The traditional method for dealing with data streams is a record-at-a-time, using either an upstream backup or replication for fault tolerance, both of which are costly. Batch processing is both efficient and fault tolerant, but increases latency.

Matei introduced their solution, discretized streams, or D-streams. They split up the input stream into chunks for MapReduce processing, and can feed back results into the next data set. Since they implement D-streams on top of Spark, they use Resilient Distributed Datasets (RDDs) and logs of translations in place of replication or upstream backups. In tests, they found they could process 2 GBps or 20 million records per second using 50 nodes, and recover from a node failure in less than one second.

Batching data in small timesteps enables both efficient parallel processing as well as parallel recovery schemes. And by leveraging Spark, they provide a functional programming interface.

Marco Serfini (Yahoo! Research) pointed out that upstream backup, which appears similar to their approach, is bad for time to recover. And good stream processing needs to tolerate load spikes. Marco wondered whether they tried to evaluate this? Matei said that their system can recover while still processing streaming data, but it can take a long time.

Using R for Iterative and Incremental Processing

Shivaram Venkataraman, UC Berkeley; Indrajit Roy, Alvin AuYoung, and Robert S. Schreiber, HP Labs

Shivaram Venkataraman presented an alternative approach to stream processing from the previous paper. Shivaram presented Presto, a prototype that extends R, an array-based language, that runs on clusters and supports incremental processing. Shivaram asserted that many problems processed today using MapReduce could be better handled using iterative linear algebra operations.

Shivaram said that their project faces many challenges. R itself has over 2000 packages for analysis, and is single-threaded for running on a single machine. He also discussed having to develop a storage driver—Presto uses HBase or HP Vertica—as well as needing memory management, partitioning, and scheduling. For fault tolerance, Presto uses primary-backup replication of the master node, and can restart workers to reconstruct lost partitions, somewhat like Spark.

Derek Murray asked if they used a dense matrix. Shivaram replied that they specify if a matrix is dense or sparse. Apache Hadoop, for example, uses a sparse matrix. Amal Fahad of the University of Rochester wondered how they decided to partition their data. Shivaram said they use a configurable threshold, and see some convergence after some number of iterations.

The Resource-as-a-Service (RaaS) Cloud

Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir, Technion—Israel Institute of Technology

Dilma Da Silva, IBM Watson Research, presented this work, as both of the Ben-Yehudas were unable to attend the conference because of a last-minute family issue. Dilma began by pointing out that with IaaS, there is a trend toward shrinking the duration of rental periods, with more fine-grained resources offered for sale. Along with provisioning according to service level agreements (SLAs), these trends will drive IaaS toward RaaS. Already we have CloudSigma and a spot market for EC2 instances, where you can specify how much you are willing to pay for a VM or an SLA.

In the paper, the authors project that client pressure for efficiency will drive cloud providers to supply levels of quality of service: prices for reaching a 95% level of some quality of service (QoS) versus reaching just 80% of the QoS over time. The provider can sell the 95% level for more, and sell spare cycles to those with less concern about QoS. To deal with this fluctuating market, agents will be needed both to set pricing and to acquire resources. The result will be a real market.

Discussion

A very active discussion followed the three presentations. Byung-Gun Chun, Yahoo!, asked the first two presenters to choose between Spark and Presto, laying out the pros and cons of each. Shivaram said that you should pick the language you are most comfortable with. Matei agreed, adding that they are adding more interfaces to Spark, so they can support other languages, like SQL.

Malte Schwarzkopf, University of Cambridge Computer Laboratory, wondered if people should trust cloud providers to provide accurate accounting of their resources. They could lie or produce very coarse-grained accounting. Dilma responded that CloudSigma provides a dashboard that supplies accounting, and you can also use another dashboard. Amazon provides more than one dashboard, so you can choose. You can also use the power of the market and use another cloud provider if you feel you are getting your SLAs met, regardless of the accounting data provided. Someone pointed out that using automated agents can result in markets with violent price swings. Dilma agreed, saying that she had looked at Amazon spot prices, and they already vary a lot, but that this is an understood issue.

Tim Wood, University of Washington, wondered that with the high cost of fault tolerance, is it possible to have projects that don't care about fault tolerance. Matei responded that he suspects that some projects, like machine learning, don't require that property. But having fault tolerance makes it easier for programmers to imagine how things work. Tim Wood commented that you can pay more for a more precise answer. Derek Murray of MS Research asked Shivaram whether they need to serialize the records determinantly to make consistency work. Matei responded that the data is a set, and you need to decide what's in it. Shivaram said their approach uses functions for partitioning.

Raja Sambasivan wondered if an IaaS client that relied on many downstream services would need to have SLAs for those downstream services. Dilma answered that you can specify the resources required for each level of your application, using the same system described in the paper. Someone else asked Dilma if she or the authors were foreseeing a significantly different OS or programming system instead of Linux and Windows with all of their libraries. Dilma said there is motivation for building a new OS and runtime environment that can handle more flexibility. There are a few groups exploring resources as a market, and they are considering moving away from UNIX APIs and processes.

Yun Mao of AT&T Labs asked what happens if the data source reaches a higher rate than actual processing. Matei responded that if the data includes a timestamp, this can be used to update the state. But this must be done at the application level. His group would like to add libraries that support doing this and other things. Andrew Wong of UCB asked if

they had compared their research with MPI. Shivaram said that programming in MPI brings impressive gains in performance, while losing in programming expressiveness. Andrew said that he had seen some high-level abstractions on top of MPI, as well as seeing an order of magnitude improvement over things in their project. And he can use Python. Shivaram responded that Presto is Python powered. Byung-Gun Chun said that if he can checkpoint and batch, what did their software add? Matei answered that they are already doing more than just batching in their handling of data streams.

Mike Kozuch, Intel Labs Pittsburgh, the session chair, asked an open-ended question about programming models vs. languages. Being a programmer relies on the ability to pop between different languages, but the cloud tries to hide this from their customers. Will people have to be aware of the underlying issues in their use of clouds? Matei responded that with Spark they are trying to see the limits of what can be done with an execution engine. The engine provides a deterministic graph, and a lot of parallel algorithms can be reduced to this low-level computation engine. At the top, you exchange queries at a high level. He finds it interesting to ask about the limits of this type of model and about what needs to change to support other models. Shivaram answered that the challenge is how composable some of the properties are; can you express this in one sequence of operations? It is challenging to come up with examples that compare Python to Scala, the basis of Spark. Dilma provided a final comment: we in the systems community look at this from the perspective of performance, but in HPC, programming with MPI is very expensive; we have to be careful not to go in that direction.

Posters

Summarized by Dan Levin (dlevin@net.t-labs.tu-berlin.de) and Phil Schmidt (phils@net.t-labs.tu-berlin.de)

The HotCloud '12 poster session included approximately 10 posters, some of which had been presented as full or short papers during the workshop.

Saving Cash by Using Less Cache

Timothy Zhu, Anshul Gandhi, Mor Harchol-Balter, Michael Kozuch CMU, Intel Labs

Applications which rely on database back-ends often make use of simple key-value caching mechanisms (e.g., memcached) to improve performance by reducing load on the database. Dedicated machines are often provisioned to act as transparent cache instances, but provisioning and running these instances incur an operational cost. This work presents an analysis into the feasibility of achieving sufficient database offloading while minimizing the number of provisioned cache instances subject to performance goals. The authors leverage the Zipf-like distribution of database content in different workloads to explore the potential for cost savings while maintaining performance. The authors also consider the impact of the ratio between peak and average load properties of different workloads on the potential for cache and operational cost reduction.

vCrib: Virtualized Rule Management in the Cloud

Masoud Moshref, Minlan Yu, Abhishek Sharma, Ramesch Govindan, USC Viterbi School of Engineering

Managing forwarding state in the context of virtualized cloud environments is difficult. Network forwarding state changes in response to changes in network policy (e.g., access control), host mobility, and responses to routing updates and failures. Furthermore, forwarding state must be managed both at the VM hypervisor as well as at the forwarding devices within the network—these two areas expose different strengths and limitations in terms of what forwarding state can be expressed and what performance can be realized. This work, which follows up ideas from the "DIFANE" Sigcomm paper, attempts to address the challenges in managing the partitioning, placement, and updates to distributed, heterogeneous forwarding state.

How Much Energy Can You Save? An Energy Perspective of YouTube

Zhonghong Ou, Hao Zhuang, Antti Ylä-Jääski, Aalto University, Finland

Given the limited battery resources of mobile computing devices, reducing power consumption is of utmost importance. This work presents a preliminary analysis of power consumption on mobile devices in the context of downloading and watching YouTube streaming video. When mobile devices play streaming video, they incur both a cost in terms of radio power consumption as well as video decoding. When video streaming traffic leads to prolonged radio transmission, power consumption on the mobile device can be unduly increased. Alternatively, faster bursting with aggressive buffering of traffic may lead to improved power consumption at the mobile device, as the radio may be turned off for longer periods. The authors compared two different YouTube video clients which employ different video buffering to allow more or less aggressive download bursts. The measured power consumption is compared between these two clients to argue that power savings can come through optimization of traffic streaming behavior.

Automated Diagnosis without Predictability Is a Recipe for Failure

Raja Sambasivan, Greg Ganger, Carnegie Mellon University

Performance analysis on complex distributed systems is becoming more and more important, but the variance in performance data of such composed systems makes it sometimes hard to reason about performance problems. This work presents a nomenclature about how to reason about performance data variance in composed systems and tools to deal with this variance. The authors categorize sources of

variance as "intentional variance," introduced intentionally by system designers, i.e., through randomized algorithms; "inadvertent variance" by, for example, poor quality code; and "intrinsic variance," which is part of the system domain and cannot be removed anyway. The tools they intend to build should help to find sources of variance in the system by analyzing traces from all levels of a system and estimating where the variances in the performance data originate.

Overcoming Risks in Hidden Dependencies within the Cloud

Ennan Zhai, David Isaac Wollinsky, Bryan Ford, Yale University

If services within the cloud are composed of various cloud services, there is a strong temptation to use similar services of different cloud providers for redundancy. But are these services really independent? To give guarantees and calculate failure probabilities, it is crucial to know that, but cloud providers will keep this information as a business secret. To solve this problem, the work proposes introducing a trusted third party—the Cloud Reliability Recommender—which is entrusted with the hidden dependencies and their reliability information and therefore can do correct fault tree analysis and give correct fault probabilities for given configurations.

Towards Fair Sharing of Block Storage in a Multi-Tenant Cloud

Xing Lin, *University of Utah;* Yun Mao, *AT&T Labs—Research;* Feifei Li and Robert Ricci, *University of Utah*

While it is advisable from a cost perspective to share storage resources between as many applications as possible, their workloads might negatively impact one another. In their analysis, the authors show that random read workloads negatively impact all sequential workloads, and random write workloads negatively impact all workloads. They propose a storage architecture they call *FAST* using three redundant copies. Random and sequential reads are sent to two different storages where all write effects are neglected by caching. The third storage uses a log-structured storage to provide data safety even in case of system failures and which does not suffer from random writes.

Working with Big Data

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

Big Data Platforms as a Service: Challenges and Approach

James Horey, Edmon Begoli, Raghul Gunasekaran, Seung-Hwan Lim, and James Nutaro, Oak Ridge National Laboratory

James Horey kicked off this talk by briefly describing the "big data" work done at the Oak Ridge National Lab (ORNL). A lot of their work happens in collaboration with, and is sponsored by, federal agencies, who have many large data sets of variable size and nature. As a result of these variable needs and due to its flexibility, cloud computing is a highly interesting paradigm for the ORNL. Nonetheless, the existing paradigms of infrastructure, platform or software-as-a-service (IaaS/PaaS/SaaS) are not exactly what the sponsors require: instead, they want "Big Data Analytics as a Service," which amounts to using familiar, standard methods and commands to set up cloud environments and execute analytics jobs. Indeed, Horey said that what the ORNL scientists identified as a key need is a "cloud package manager"—an analogy to Debian's "apt-get"—to set up and configure complex distributed systems, respecting interdependencies.

To satisfy this need, Horey et al. implemented "Cloud-Get," a concept for an apt-like repository for cloud services such as Hadoop, Cassandra, etc. By way of example, he discussed the command "Cloud-Get install cassandra --nodes=20 --storage=20TB", which would configure a 20-node Cassandra cluster with 20 TB of storage. The aim here is to achieve standardization of deployment procedures, and to provide a common infrastructure that package writers can hook into. Furthermore, an additional goal is to support elasticity, i.e., "reconfiguration" of packages, for example with different numbers of nodes. Finally, moving data between systems is hard today, and requires ad hoc scripts. Instead, Horey proposed a tool dubbed "data-get," standardizing the process of moving data packages, akin to a UNIX-named pipe. The system does, however, rely on an external cloud manager (in the prototype, Xen and ClockStack are used). Service package authors must then define VM types, as well as dependencies between classes and event handlers. This abstraction operates a level above IaaS: users do not need to know about VM or storage placement, and the package manager does not necessarily control scheduling and co-location; it can make QoS guarantees.

Someone asked how the ORNL toolchain compares to open source system configuration tools like Puppet. Horey replied that these tools operate on a different level, setting up bare VMs rather than entire distributed systems. The Cloud-Get suite makes it possible for non-expert users to easily configure complex systems. Another questioner pointed out that cloud tools like Hadoop are hard to configure, and setting them up surely cannot be as simple as just using Cloud-Get, as they still require user configuration? Horey answered that, indeed, performance-tuning systems are hard, and not solved by Cloud-Get. However, he pointed out that its goal is to have a quick way of setting prototype systems up, which can then be tuned further.

Why Let Resources Idle? Aggressive Cloning of Jobs with Dolly

Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica, University of California, Berkeley

In this talk, Ganesh Ananthanarayanan presented Dolly, a system to speed up the completion time of MapReduce jobs

by opportunistically duplicating them. The large fraction of relatively small, but interactive, latency-constrained data-analysis jobs observed, for example, in traces from Facebook, are not only especially sensitive to stragglers, but also particularly benefit from accelerated completion. Previous approaches to straggler mitigation, such as blacklisting bad machines, do not mitigate non-deterministic stragglers. Speculative execution of backup tasks may help with this, but making the decision to speculate is tricky—especially with small jobs, which do not run long enough to take many samples of their progress, and which often execute all of their tasks in parallel, achieving no amortization of stragglers over time. Existing approaches only really help with large jobs; small jobs still have 6–8x difference in runtime between the median and the slowest task.

In contrast, the authors propose to proactively launch entire job clones, and then pick results from the earliest clone to finish, thus probabilistically mitigating stragglers. Of course, a key question is whether there are sufficient resources to do this. Most clusters are underutilized due to provisioning for peak load, so the outlook is hopeful. Of course, cloning could still lead to a "tragedy of the commons," where everyone is using extra resources on the assumption of low utilization. But, as Ananthanarayanan pointed out, in the case of Facebook, 90% of jobs only use 6% of resources, thus making it entirely feasible to duplicate these small jobs at a modest utilization impact. One challenge, however, is that well over three clones are needed in order to statistically avoid stragglers. Having this many clones leads to contention on input data with naive cloning at job-level granularity. Instead, the authors propose to still clone the entire job, but do so at the task-level, and pick the first task to succeed out of every equivalence class. This means far fewer clones are needed, although Ananthanarayanan pointed out that contention on the intermediate data, e.g., from map outputs used by all reduce task clones, is as of yet an unresolved issue. When evaluating Dolly against the baseline of LATE, using a trace-driven simulator, on a month-long Facebook trace and with a 5% cloning budget, the authors found a ~42% reduction in completion time on small (<10 tasks) jobs for task-level cloning.

A member of the audience pointed out that the size of the small jobs' input data is likely to be very small, which Ananthanarayanan confirmed, saying that it is usually tens or hundreds of megabytes. The questioner then asked why one would even bother with distributed parallelism for such jobs. Ananthanarayanan explained that small jobs often share inputs with larger jobs, the latter point necessitating running on a cluster.

Another questioner asked if any thought had been given to input data size skew as a source of stragglers in Dolly. Ananthanarayanan said that their slowdown measurements took input size into account, but that such deterministic stragglers were not otherwise considered. Finally, it was pointed out that Dolly clearly assumes fully deterministic tasks, and non-determinism could cause trouble for the cloning strategy, which Ananthanarayanan confirmed.

Predicting Execution Bottlenecks in Map-Reduce Clusters

Edward Bortnikov, Yahoo! Labs, Haifa, Israel; Ari Frank, Affectivon Inc. Kiryat Tivon, Israel; Eshcar Hillel, Yahoo! Labs, Haifa, Israel; Sriram Rao, Yahoo! Labs, Santa Clara, US

Alex Shraer explained that this work takes a motivation similar to Dolly: stragglers are bottlenecks in MapReduce clusters. Commonly, people use the dual strategies of avoidance (reduce the probability of straggler occurrence, for example, by using data locality) and detection (identification of stragglers, followed by speculative backup tasks) to deal with them. However, all existing approaches are based on heuristics. In this work, the authors use machine learning to predict stragglers. The main motivation for this heavy-weight approach is that speculation is typically quite wasteful: 90% of speculative tasks at Yahoo! are killed because the original ended up finishing first. Looking at the 15% top straggler nodes over time, they find that there are some consistently pathological nodes. Furthermore, jobs are often re-run many times: 95% of mappers and reducers are part of jobs that run more than 50 times over five months. This suggests that historical information can be used as a predictor of stragglers.

To do so, the authors look at the task slowdown factor, the ratio between a task's runtime and the median runtime among sibling tasks in the same job. They find the root causes of stragglers to be data skew (rarely >4x, though), and hotspots (hardware issues, contention). A sample over ~50k jobs shows that mappers with a slowdown exceeding 5x occurred in 1% of jobs overall, and in 5% among of jobs with more than 1000 mappers. Of these slowdowns, 60% were due to skew, and 40% due to hotspots. With reducers, slowdown is even more common: 5% of overall jobs experience it, and 50% (!) of large jobs do (10% of these due to skew, 90% due to hotspots).

The authors propose developing a slowdown predictor ("oracle") to address this. As inputs, it takes machine and task features, and produces a slowdown estimate as its output. Their prototype predictor considers a large number of features, detailed in the paper. For slowdown prediction of mappers, they find an R^2 value of 0.79; for reducers, R^2 = 0.401 (where 1.0 is optimal). Hence, map stragglers are better predicted than reduce stragglers; Shraer suggested that one might be able to alleviate this difference by using stage-specific predictors. He also pointed out that this work

focuses on the "average straggler" rather than the pathological outlier.

There were no questions after the talk.

Discussion

In a lively panel session, the discussion revolved around ways of achieving better predictability for large-scale data processing, as well as the nature and quality of the data itself. The first questioner asked if there was any scope to be more creative in Dolly: for example, sub-splitting tasks to achieve more parallelism, or processing the data from opposite ends until the tasks "meet." Ganesh Ananthanarayanan replied that Dolly makes the assumption that the task size is already optimized, and thus the point is an orthogonal one. Someone asked what the panelists considered to be the most challenging problems working with "big data." James Horey said that, while not a technical challenge, the ORNL scientists found that data sets were often noisy, necessitating complex preprocessing. Ganesh Ananthanarayanan suggested increased resilience to partial and total failures of tasks (stragglers and "lost" tasks, respectively) as a challenge. Alex Shraer pointed out that business intelligence of many companies depends on machine-learning algorithms, which are hard to parallelize using MapReduce, and suggested looking at this.

James Horey also made the point that many people dealing with "big data" are not necessarily thinking of using "the cloud"; instead, they often have highly specialized software loaded with domain-specific assumptions that they want to use.

Someone pointed out that the second and third presentation of the session were essentially trying to address the same problem, but using different approaches. As a result, the questioner wondered how much use of prediction might be adequate for straggler detection. Ganesh Ananthanarayanan believed that heuristics and prediction could happily co-exist: for large jobs, prediction and speculation were important, but for small jobs, cloning was affordable. Alex Shraer agreed, and emphasized that quick prediction algorithms facilitate the use of machine learning and similar approaches.

Dilma Da Silva asked James Horey what pricing model he imagined when offering big data analysis as a service, and if it differed from the currently dominant machine-hour model. Horey said that the answer was still unclear, and pointed out that users often have no idea how their analysis task translates into CPU-hours. Instead, he suggested investigating a charging model based on the amount of data analyzed.

Raja Sambasivan from CMU asked if the speakers could point out changes to frameworks like Hadoop, or indeed to hardware, that one could make to achieve better or easier predictions. Alex Shraer saw two main problems: first, how to collect information efficiently, given that interactive log analysis (as used in the paper he presented) does not scale to a live production system; and second, that it is unclear what the best time to do the prediction is: before or during scheduling, or at task runtime. Changes to systems could help facilitating answers to these questions. Ganesh Ananthanarayanan concluded the panel session by emphasizing that there is a lot of heterogeneity and variance at low levels (hardware), and that ways to extract more fine-grained profiling data would be useful to high-level system optimization.

Scheduling

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

Dynamic Virtual Machine Scheduling in Clouds for Architectural Shared Resources

Jeongseob Ahn, Changdae Kim, and Jaeung Han, KAIST; Young-ri Choi, UNIST; Jaehyuk Huh, KAIST

In a virtualized environment, VMs share physical machine resources. Contention on these resources can lead to performance degradation, and the multicore trend increases the degree of resource sharing. The authors assert that cache contention is especially a problem: cache coherence protocols mean that excessive cache misses on one core can cause a cascade of cache evictions on others. Previous work tries to mitigate this on a purely intra-machine level by balancing threads to minimize system-wide miss rate and interference in shared caches. This does not, however, help in a cloud setting, where many similar tasks may be running on a machine. Furthermore, NUMA makes such cache-aware scheduling harder, since some memory requests will go to remote memory controllers, making them very expensive. However, virtualization technology and the use of VMs in the cloud open up a new opportunity for load balancing at the global level using live migration (e.g., of customer VMs in a public cloud). The goal is to try and minimize global LLC miss count and/or global remote page access count. Ahn et al. compare the best case placement of 32 VMs running different workloads from SPEC CPU to the worst-case setup, as well as an interleaved memory allocation setup, where pages are allocated both on local and remote NUMA nodes. They found that there are significant benefits over the worst case to be had from placing VMs in a micro architecture-aware manner.

Based on these findings, the authors propose a system with a global cloud manager node, which makes global scheduling and re-balancing decisions in cache-aware and/or NUMA-aware fashion. Back-end nodes report their statistics to this node, which then makes decisions to move VMs between sockets inside a machine, and across machines in the cluster, in "local" and "global" phases, respectively.

In their experiments, the authors used a four-node testbed (plus one manager node), with each node having eight cores across two sockets with 12 MB shared last-level cache. They ran eight VMs using 1 GB memory and 1 vCPU on each machine, and find that, for different workload mixes, the cache-aware and NUMA-aware schedulers can give significant improvements over the worst case when combining CPU-bound with memory-bound workloads, after taking VM migration costs into account. With homogeneous workloads (e.g., all CPU-bound), the improvement is far less substantial.

There were no questions after the talk.

North by Northwest: Infrastructure Agnostic and Datastore Agnostic Live Migration of Private Cloud Platforms

Navraj Chohan, Anand Gupta, Chris Bunch, Sujay Sundaram, and Chandra Krintz, University of California, Santa Barbara

Navraj Chohan started this talk off by explaining how private PaaS offerings can bring cloud technology (elasticity, distribution, fault tolerance, high availability) into the local environment and thus enable programmer productivity, as programmers no longer have to worry about distracting setup and maintenance issues which are not part of the core application. He also introduced AppScale, a PaaS offering similar to Google AppEngine, but which is infrastructure- and datastore-agnostic (unlike AppEngine). This is achieved by having abstraction layers, allowing high-level use of GQL and Google's datastore API, as well as ZooKeeper for coordination and transactional semantics, while supporting a range of different back-end stores.

The reason why such support is crucial, according to Chohan, lies in the fact that PaaS systems are being upgraded all the time—both at the underlying hardware layers and at OS-level and datastore software layers. Ideally, this should be possible with minimal downtime and overhead; furthermore, it is useful to be able to substitute another datastore underneath the system, eliminating vendor lock-in, and facilitating seamless upgrades to better technology. AppScale achieves this in a real-system, with backwards compatibility and transactional semantics, and without data loss. Chohan explained an example: moving a storage solution from Cassandra on OpenStack to HBase on Eucalyptus. This is an automated six-step process in AppScale: (1) the new deployment is initialized; (2) ZooKeeper metadata is synchronized; (3) a temporary memcache is provisioned and warmed up (in order to minimize load on storage backend during the migration); (4) a snapshot of the datastore is initiated, transferred over, and loaded into the new system; (5) data proxy mode is entered in order to keep the old system running to finish off any outstanding requests; (6) and, finally, handover to the new system occurs.

Chohal showed some evaluation results from a single-node deployment, moving from Cassandra to Hypertable. The authors found that the per-request overhead is less than 1% during migration, because the memcache is very fast. The ZooKeeper metadata synchronization takes about 45 seconds for 100,000 locks, and the overall difference in client latency between normal operation and the background migration situation is small and well within acceptable bounds.

There were no questions after the talk.

Automated Diagnosis Without Predictability Is a Recipe for Failure

Raja R. Sambasivan and Gregory R. Ganger, Carnegie Mellon University

This provocative talk by Raja Sambasivan began with the oft-quoted statement that systems research is all about tradeoffs. He pointed out that there are a few commonly agreed upon metrics: for example, correctness, performance, reliability, and power. He, however, argued that a key metric has been ignored, but is very important: predictability. Indeed, Sambasivan explained, this is especially true for distributed systems, as it affects the ability to optimize other metrics. And what is worse, there is also no single fix or answer—achieving predictability requires lots of hard work when building systems. Above all, low performance variance is essential, but sometimes hard to achieve.

However, without predictability, resources are wasted (because the slowest resource dominates), and also providing SLAs becomes much harder. Sambasivan pointed to this as his key motivation for this work, and it is also important for the success of automated diagnosis. There exist many automated diagnostics tools, but few are making it to production use. Sambasivan claimed that the problem is not the tools but the system, or rather, the layering of many systems on top of each other. Since diagnosis tools focus on deviations in metrics to localize performance problems, high variance makes their life very difficult. Sambasivan exemplified this using the sample distributions of two metrics: if they do not overlap very much, they probably represent different underlying distributions. But if they do, because of high variance, we cannot really tell. This is complicated even within a single system, but with a complex setup of interacting systems, a diagnosis tool's life is incredibly hard. The key takeaway of this point was that the usefulness of a diagnosis tool is really limited unless the entire system has good predictability.

Unfortunately, as Sambasivan re-emphasized, there exists no secret sauce here, just hard work. Developers must identify sources of high variance, and rigorously isolate them. The authors thus postulate the "three I's" of variance: inadvertent variance is unintentional, but can be reduced; intrinsic variance is fundamental (e.g., disk performance), so it must be isolated; intentional variance is a result of a tradeoff made by developers (e.g., low-latency scheduling) and must be isolated. Still, Sambasivan pointed out, there remain many

open questions: how much reduction in variance do we need to achieve predictability properties? If intrinsic variance is significant, maybe we need to change something about our hardware to reduce it? If intentional variance is significant, maybe we need to re-evaluate our tradeoffs?

There were no questions after the talk.

Discussion

The panel discussion for this session consisted partly of questions relating to Navraj Chohan's talk about live migration in AppScale, and partly of discussion on Raja Sambasivan's talk about predictability and variance in large systems. On the former topic, someone asked whether the migration assumed that, apart from the ZooKeeper locks and the actual datastore, the system being migrated was stateless. Chohan confirmed that this is the case, and pointed out that most APIs in this space are stateless. In a second question, Chohan was asked how the switch between two completely different databases is performed, and what translation is applied to the data. He explained that datastore is dumped as a set of blobs, which are converted to protocol buffers, and reinserted into the new datastore one entity at a time. Furthermore, he said that some cleverness to deal with locks is required, making sure they are mapped correctly in a different system.

In the discussion pertaining to Raja Sambasivan's presentation, an audience member commented that he would go even further than postulating a need for tools to cope with variance, since a lot of new code is built with every system, introducing new sources of variance and uncertainty. Sambasivan responded that this is indeed a tough challenge. He suggested that we might be able to annotate libraries, or evolve a policy of flagging some operations as having (potentially) high variance in the documentation. Another questioner inquired whether the way to build better systems would be to probe lots of different metrics, but consensus was that even then, variance would still occur and present a challenge. Finally, someone asked, somewhat provocatively, whether we really have such an essential need for predictability as the talk made out, noting that we do have systems that deal with unpredictable events. Sambasivan said that he did not have an answer either, and that it might be interesting to further investigate building predictable systems from unpredictable components.