12th Workshop on Hot Topics in Operating Systems (HotOS XII)

Monte Verità, Switzerland
May 18–20, 2009

KEYNOTE ADDRESS

- ***The Elements of Networked Urbanism***
  Adam Greenfield, Head of Design Direction, Nokia

  Summarized by Simon Peter (simon.peter@inf.ethz.ch) and Tudor Salomie (tsalomie@inf.ethz.ch)

Adam is working on a book called *The City Is Here for You to Use* and his talk was related to that. Adam began with a speculative manifesto and a diagnosis on where converging technical and social possibilities in our environment are taking civilization. If the promises of ubiquitous computing came true, how would we be living?

Over 50% of the world's population is now living in cities, and this trend is accelerating. Today's mega-cities are prototypes of the conditions within which post-urban humanity is going to live in. On the other hand, there are de-populating cities, like Detroit, that are beginning to lack vital infrastructure, like police and fire-fighters.

By the end of 2012, embedded network sensors will be responsible for 20% of non-video Internet traffic. By then the Internet will no longer be primarily a human-to-human communication channel. Instead, an increasing amount of data about the physical environment will be exchanged. Due to these factors, technology will be intersecting primarily with an urban population, not civilization in general.

Adam structured his talk into 14 rough transitions that are likely to develop in urban societies:

1. Networked resources will be the components of urban environments. We will be surrounded by physical installations that have IP addresses and are probably programmable, afforded by IPv6.

2. Open APIs will become lingua franca. Consumers will be plugging systems seamlessly into one another. Moore's Law has given us cheap, powerful sensors, and we are getting to a point where we just incorporate them anywhere because they are so cheap.

3. Building blocks of our cities will be able to adapt to changing conditions. Buildings will be able to configure

themselves in real-time to conditions of load, weather, utility, usage, etc. Large structures will be able to move, shift, and adapt.

4. Latent quantities become explicit and abstractions grow teeth as data generated by sensors is processed and visualized in real-time. People's decisions and actions will be impacted much more by abstract quantities, such as restaurant health inspections, air quality, and crime rates, than today, providing a power shift in favor of citizens.

5. We will transition from browse to search urbanism. Today, we browse based on our senses in the area we live. In the future, we will be able to query the environment as everything becomes networked. This consolidates our natural desire for homogeneous communities. We will be looking for things and people we are already comfortable with, ignoring anything else. According to Adam, this will have negative effects where it impacts democracy.

6. Instead of holding information, we will be sharing information much more than today, as the cost of sharing drops to almost zero.

7. We have built our culture on the expectation that information eventually expires. An artifact of the networked condition is that information tends to persist. For example, the criminal record of juvenile offenders would typically be expunged from the records. In the networked world, such information is much more likely to persist.

Someone asked whether falsehood will persist just as much as truth. Adam agreed. As statements will likely be decontextualized when processed, their truth value will be much harder to assess, even though there may be networked ways around that, like distributed reputation databases and reputation economies.

8. The transformation of a city from passive to interactive has already begun, as exemplified by buildings whose facades are transformed into active displays. Still, Greenfield considers these dull and passive; true interactivity is only achieved when one can push/turn/change the way things look. He envisions the entire fabric of a city becoming interactive at a more fundamental level.

9. Another transition that Adam talked about is that from way-finding to way-showing. The problem at hand is that of going from point A to some other point, as described by Kevin Lynch in *The Image of the City*. Currently, people know how to navigate through a city, but with the appearance of the new dimension, that of knowing one's exact position, cartography and orientation change. Context-based orientation leads us away from way-finding to way-showing (envision the sidewalk lighting up just for us in order to show us the way). The positive aspect is that it removes the problem of getting lost, while the negative aspect is that it eliminates serendipity. Greenfield also pointed out how fallible such systems can be.

10. All objects will evolve into services. Adam sees the physical object as realizing its full potential only when it becomes a networked object. For example, his motorbike, only used 20% of the time, could reach a higher degree of utilization if it were shareable and bookable (transforming it into a service). The issue that is observed is that when an object becomes a service it will not morph, but it will be very hard to anticipate what it can actually do.

11. We should stop thinking about vehicles and more about mobility services. Every trip is going to involve walking, private vehicles, shared private vehicles, and public vehicles. These networked services will allow you to build your agenda and itineraries using them as resources offered by mobility services.

12. Adam underlined the next transition as very important from his perspective. He talked about ownership and use. In contrast to owning music, online services provide access to music libraries at minor costs (listening to commercials every so many minutes). It undermines the current economic model, as goods become nonrival (they can be used simultaneously by multiple consumers).

13. When talking about the transition from community to a social network, Adam began by trying to express what is meant by community. Subconsciously, a community sees itself as a network. He wondered whether in this case we are the nodes of such a network, but he could not give an answer. He is capable of envisioning what networking means for things such as blocks or buses, but not for people. The second topic he touched on in the context of this transition was that of the FOAF (friend-of-a-friend) specifications. Such specifications only allow neutral or positive characterizations. Adam disagreed with this and countered that in order to define ourselves we must be able to say what we are not, as well as what we are.

14. The final transition goes from consumer to constituent. We have learned how to consume goods, services, and experience. Adam hopes that, based on all the transitions he mentioned, we shall all become more active producers and take a greater role in transforming the world.

Adam concluded by saying he cannot foresee all the impacts of networked urbanism and he leaves this as an open question. He said that the people designing systems had no clue that things would change when you connect them!

Jorrit Herder asked about the technical challenges involved in accelerating or decelerating these transitions. Adam replied that there are no technical challenges; the challenges are in the openness of standards, systems, or APIs, which would lead to lower costs of understanding and connection.

Michael Scott, considering the final transition from consumer to constituent, worried that technology would concentrate the power and the money even more, rather than democratize it. He argued his case using the example of pay phones, which are dying out. Adam agreed that a small number of nodes will concentrate a lot of power within the urban network, and he also pointed out the digital divide, in which rich people will be able to "hide"

from the network, while the poorer will have to rely on the network. Following up on this idea, Michael asked whether everything will be accessible as long as you pay for it. Adam clarified: you will pay through loss of privacy, not with money. His example is that it becomes impossible to refuse connectivity, as sometimes the social incentives to it are too powerful.

Tim Brecht asked whether the cities mentioned are poor cities and therefore have less access to technology. Adam cited the rural area of Chengdu (Sichuan, China), where the penetration curve of mobile phones is extremely high, offering an incredible platform for networking and ubiquitous computing. He added that only a couple of years ago half of the world's population had yet to make their first phone call, while today it is down to the last billion.

Michael Kozuch said he understood the network part of the talk but was unclear what was so special about the urbanism. Adam answered that the human species is becoming an urban species. He classified locations into urban areas (characterized by a high density of nodes with a lot of aggregation possibilities), suburban areas (in which conditions for connectivity exist), and rural areas (in which a push factor is required for the network to come to life). Adam sees suburbanization within the urban as creating homogeneous groups within the urban environment.

IT'S DEAD, JIM

Summarized by Adrian Schüpbach (scadrian@inf.ethz.ch)

- ***Hierarchical File Systems Are Dead***
  Margo Seltzer and Nicholas Murphy, Harvard School of Engineering and Applied Sciences

Margo explained that browsing is increasingly transitioning to search. She claims that many file systems are dead now. Namespaces are hierarchies, she explained, but real people's view of namespaces is search. So what should be done? Files are objects with different attributes, and a decision has to be made where they should be stored. Deciding that depends on the creation of the namespace. It also means designating a most important attribute, the one where the hierarchical name starts.

Margo noted that we have to know something about an object to be able to find it; the problem is that we have to organize the physical world and model it as a virtual world. Filing cabinets, for example, may be used for papers and organized by author. The problem here is that there is only one physical object, which leads to serious constraints. But taking this model to the virtual world releases some constraints, because objects can, for example, be duplicated, if the amount of data is not too big. Though we often have too much data for duplicating, we can use database systems to manage and query large amounts of data efficiently. Database systems are sometimes too heavyweight or too expensive, however, so the "poor man's" data management is done using a file system.

Margo proposes a new architecture that would eliminate the hierarchy as structuring mechanism. This new architecture consists of stable storage, an object index, and metadata. On top of this, type-specific indexes, like POSIX names, and full-text or image search can be implemented. Rather than implementing and indexing on top of POSIX, Margo and her group are implementing this architecture because POSIX is too limiting and it could be simpler to start from scratch.

Steve Candea pointed out that not every document has attributes or tags assigned by users, and users might not remember where the document is after a time. Margo replied that by using indexes it might be easier to find documents even after a long time. Someone wondered if she was comparing a file system to the Internet. Margo replied they are not trying to compete with the Internet, but users do not need to know where their data is stored, just how to access it. How did they control access to objects in their approach? In file systems this is done by access bits on directories. Margo replied that security should be done by security attributes assigned to objects rather then by performing access control on directories. Directory-based access control only makes sense if similar documents are stored in the same directory.

- ***An End to the Middle***
  Colin Dixon, Arvind Krishnamurthy, and Thomas Anderson, University of Washington

Colin said that we don't need middleboxes such as caches, traffic shapers, firewalls, NATs, VPN, proxies, and load balancers: we only need the functionality these boxes provide. He said the reason why we are using these boxes is that they are convenient, but they are expensive. For example, a Cisco box costs $3000–$4000, so companies spend a lot of money on these boxes.

He noted that large networks today are usually managed via a diverse set of proprietary hardware middleboxes with mixed interoperability, and small and home networks are usually built with unmanaged low-cost routers which do almost nothing. Unlike companies, home networks don't need high performance, but they do need a reliable network all or almost all the time.

To make the management of these networks more efficient for the users, he proposes a new approach. In their approach, the network services run in specialized attested VMs, which is an attested execution environment. Currently, this is a lightweight Linux VM. Colin says that distributed systems are complicated, especially because some types of networks are not reachable or too expensive, but he still wants to tackle the problems.

Armando Fox said that the problem is that these middleboxes are not commoditized, but they should be. If you have to trust a chain of VMs that run network services, someone wondered, why not trust a Cisco router? Colin answered that in our architecture there are only VMs with shared hardware resources.

- ***No Time for Asynchrony***
  Marcos K. Aguilera, Microsoft Research Silicon Valley; Michael Walfish, University College London, Stanford, University of Texas at Austin

Marcos explained the problem of node failing in distributed systems. If, for example, the primary fails, after some timeout the backup becomes the new master. However, an end-to-end timeout is hard to get right. If it is too short, there are two masters, and if it is too long, the system is unavailable for too long. Someone should attempt to build a system without timing assumptions. The conventional wisdom is to design for asynchrony; many systems have Paxos and are safe under asynchrony, but it comes with costs—algorithmic costs and hardware costs—because asynchrony requires at least three machines.

There are three different approaches to the problem: (1) keep it simple and rely on timeouts; (2) keep it safe and design for asynchrony; (3) their approach, which is that there is good in both views but both are extreme. They want simplicity, safety, high availability, and no end-to-end timeouts.

To attain this, Marcos proposes spies which indicate a crash in an authoritative way, by using local information like local time or enforcing a crash by killing a process.

Marcos argued that asynchrony is problematic in practice because higher levels often use deadlines and might decide wrongly. Safety and liveness are separable in theory but not in practice. Under asynchrony, components hide useful information. If components are not responding, higher layers have to guess why and a wrong guess leads to loss of safety. Asynchrony has a complex design which leads to mistakes and safety violations.

Marcos introduced the perfect failure detector abstraction (PFD), which always tells "up" or "crashed" for a given service with strong accuracy and completeness. They realize PFDs not by killing whole machines as current approaches do, but by taking smart decisions on what to kill. Knowledge of different layers of the local system tells the PFD whether a certain component crashed. Spies in different levels control each other. They can find the smallest crashed component. That leads to a simple, safe, and live distributed system.

Someone noted that in shooting to kill, he would need to wait for a certain time until he was sure his target was dead. Does that lead to timeouts again? Marcos responded that they rely on local timing. Margo asked how she would know that killing worked. Did you move the third Paxos machine to the switch? Armando answered that he moved the responsibility from the third Paxos machine to the switch, which gives more evidence that killing worked. Roscoe asked what the metric is for simplicity. How do you measure that a spy is less complex than Paxos? Marcos replied that they could count the number of lines of code. Someone else asked what would it cost to implement spies vs. having a guru implement Paxos. Most systems only implement something Paxos-like, not really Paxos. For spies it is easier, because they can just look at the process table and know that a process is dead.

HEADS IN THE CLOUDS

Summarized by Qin Yin (qyin@inf.ethz.ch)

- ***Computer Meteorology: Monitoring Compute Clouds***
  Lionel Litty, H. Andrés Lagar-Cavilla, and David Lie, University of Toronto

Lionel started by defining cloud computing as Iaas (Infrastructure as a service) and stating that security is the main challenge facing cloud computing. His talk focused on protecting the cloud resources from abuses, such as sending spam, hosting illegal contents or attacking other virtual machines. Other than ISP, cloud providers could use introspection to examine the VMs' behavior for signs of misbehavior.

Lionel then compared four representative introspection approaches along three axes. The four approaches are host-based agent, trap and inspect, checkpoint and rollback, and architectural monitoring. The three axes are power-defining the scope of VM events it can monitor, robustness based on the assumptions made about the monitored VM, and unintrusiveness characterizing the disturbance introduced in the monitored VM. The first approach hampers unintrusiveness, the middle two are not robust, and the last one is not as powerful. Lionel then illustrated an introspection task to determine the applications run by a customer VM and their versions. He discussed the tradeoffs among these introspection techniques and came to the conclusion that architectural introspection is promising and more research work is needed to explore the full range of events. Introspection is not a silver bullet, however, and cloud providers should be aware of its limitations.

Steven Hand asked why the spam senders will pay Amazon EC2 if botnets are free. Lionel responded that cloud is another way to send spam and spammers will even use stolen credit numbers to get Amazon resources. Garth Gibson asked whether there are ways to use introspection to assure the CIOs that the data will not be stolen or damaged after outsourcing internal applications to EC2. Lionel answered that introspection can provide assurance by checking whether the code running is known by the VM. Garth worried that CIOs may not be willing to tell what applications are running in their VMs.

- ***Wave Computing in the Cloud***
  Bingsheng He, Mao Yang, and Zhenyu Guo, Microsoft Research Asia; Rishan Chen, Microsoft Research Asia and Beijing University; Wei Lin, Bing Su, Hongyi Wang, and Lidong Zhou, Microsoft Research Asia

Bingsheng defined the cloud as large-scale data processing. The current cloud computing systems such as Google's MapReduce, Yahoo's Hadoop, and Microsoft's Dryad provide scalability, fault tolerance, and query interfaces using high-level languages. However, by examining the query trace from a production system, Bingsheng concluded that I/O

and computation efficiency of the query execution was far from ideal, because of redundant I/O on input data and common computation steps. This redundancy was caused by strong temporal and spatial correlation among queries.

Bingsheng then proposed to use the Wave model to capture the correlations. Data is modeled as a stream with periodic updates, query is the computation on the stream, and query series are recurrent queries. To wave the computation in the cloud, their system will decompose the submitted queries, combine multiple queries into a jumbo query with reduced redundancies, and enable cross-query optimization. Finally, Bingsheng presented some promising preliminary results of their ongoing project Comet, which incorporates the Wave model into DryadLINQ.

In the Q&A session, several attendees asked about the production systems and the trace in the experiment. Bingsheng explained that the trace is per-day access logs or other logs for different business units. How did they estimate the cost of the queries and choose which queries to combine into one jumbo query? The cost model can be derived from past runs and the jumbo query is constructed by examining the correlations in the queries. Matt Welsh asked about the relationship between the Wave model and multi-query optimization in conventional and streaming query optimization. They took a hybrid approach. Margo Seltzer asked whether we really need a middle point between MapReduce and parallel database. Bingsheng replied that we need database management in the cloud and cooperation between the system and database communities.

- ***On Availability of Intermediate Data in Cloud Computations***
  Steven Y. Ko, Imranul Hoque, Brian Cho, and Indranil Gupta, University of Illinois at Urbana-Champaign

Steven's talk focused on the need to treat intermediate data as a first citizen for dataflow programming frameworks in clouds. Dataflow programming consists of multiple computation stages and a set of communication patterns between them. One common characteristic of different dataflow programming frameworks is the existence of intermediate data between stages. The intermediate data is short-lived, used immediately, written once and read once; it also exhibits a distributed, large-scale, computational barrier nature. Through an experiment with Hadoop on Emulab, Steven showed that the availability of intermediate data is critical for execution, and if it's lost, current "store-locally, regenerate-when-lost" solutions will cause cascaded re-execution, which is very expensive.

Steven concluded that storage is the right abstraction—replication can stop cascaded re-execution and guarantee intermediate data availability; however, aggressive replication can cause network interference on foreground network traffic. Finally, he presented three replication policies to achieve minimal interference: replication using spare bandwidth, deadline-based replication, and cost-model replication.

Dejan Kostić asked about failure rates of existing systems. Steven gave anecdotal evidence: Google experimented with running a MapReduce job for six hours on 4000 machines and found at least one disk loss during each experiment. Cristian Zamfir asked about the window for keeping replicated data and avoiding re-execution. Steven answered that the ongoing work of deadline-based replication will replicate data every N stages and thus determine the degree of cascaded re-execution. Garth Gibson asked how the decisions will be made. Steven said that the programmer or system administrator sets the policy; in the future they will probably apply machine-learning techniques to autotune the parameter. Margo Seltzer said Stonebraker claims they can get two orders-of-magnitude better performance using a parallel DB instead of MapReduce; therefore their probability of failure is significantly reduced. The question of why not choose to use a parallel database to compute more efficiently and deal with fewer failures was left open.

SMALL IS BEAUTIFUL

No reports were provided for this session.

THINGS YOUR OS SHOULD DO . . . BUT DOESN'T

Summarized by Akhilesh Singhania (akhi@inf.ethz.ch)

- ***Migration without Virtualization***
  Michael A. Kozuch, Michael Kaminsky, and Michael P. Ryan, Intel Research Pittsburgh

Michael discussed the typical benefits of virtualization: improved communication between closely coupled workloads, migration of workloads from failing hardware, improved power management by consolidating workloads and shutting down parts of a cluster, and improved utilization of heterogeneous hardware by matching tasks to suitable machines while load balancing.

He then described the various forms of migration options traditionally used, pointing out their costs and benefits.

Process migration: where one application process is moved from one operating system to another. This approach has the benefit of migrating relatively small footprints but suffers because the migration engine needs to support a very wide interface (e.g., sockets, file descriptors, memory accesses), is very OS-specific, and generally is not used.

Virtual machine (VM) migration: where one VM image is migrated from one VMM to another. The advantages of this approach have been well studied, it is well defined, and it is widely utilized. Some drawbacks of this approach are that it continually complicates the software stack by pushing more functionality into the hypervisor to virtualize device drivers, and often the hypervisor does not expose the raw hardware interface or all the available hardware the VM image could utilize. To drive his point home, Michael showed some performance data of DPRSim2 benchmark running in various configurations. When running inside a VMM,

a significant performance degradation is observed. Steve Hand from Cambridge asked if he should expect similar performance from hardware-virtualized NICs and Michael responded, maybe lower overhead, but yes.

Obviously, Michael continued, running the OS on bare metal is a better situation, so can we then come up with some way of migrating an actual OS from one bare metal to another? The biggest challenge for this is that the OS should bind to device drivers, and when the OS is migrated, it needs to bind to the new device drivers, as the drivers will be pegged to the specific machine they are running on.

Michael now described the design space for OS migration. First there are various types of migrations possible, such as shutdown/reboot, hibernate, suspend/resume, and live migration. Then there are different locations available for migration, such as migrating to the same machine, migrating to a different machine but with identical hardware, and migrating to a different machine with different hardware. Suspend/resume and live migration are not currently supported at all. Finally, when migrating to a different machine with different hardware, the shutdown/restart method works with some support to account for new device drivers but none of the other types of migration techniques is possible. If support for live migration was added to this, all other types of migrations would be possible as well. Michael then presented a list of challenges and solutions for supporting live migration.

Michael concluded by pointing out some assumptions made. These assumptions include suggestions that the devices can be mapped to the target machine, that the OS has the necessary drivers, that devices are not visible in the user space, and that hardware attestation is available. OS migration is a valuable tool for a number of purposes but a fair bit of work is required to support it. Further, support for features like hotplugging and power management will make it easier to support it.

Steve Hand asked about the benefits of migrating like this, which would abstract away the changes in the hardware. And how does Michael propose to migrate storage (without moving tons of data around)? They use network storage, not local disks, and employ hotplug and unplug techniques. Lionel Litty asked why a VM is needed for suspend. It is not always necessary, but if a target machine is not available, then it is essential.

- ***Operating Systems Should Provide Transactions***
  Donald E. Porter and Emmett Witchel, The University of Texas at Austin

Don started with an example of how a common OS inconsistency can happen. Suppose you want to upgrade your browser plug-in. The new plug-in binary is written first, and then the browser configuration is updated to point to the new binary and new arguments. However, if the user tries to use the browser in the midst of the upgrade, or the upgrade crashes, the browser can be in an inconsistent state and various forms of corruptions can occur. What the user desires is either to have the entire installation or none at all. The POSIX API is broken.

Typically, users have simple synchronization requirements but are forced to use a fairly complex database for the tasks. This gives support for system calls in applications with transactional memory, allows fault tolerance in untrusted software modules, and atomically updates file contents and ACL. This will also make it easier to write OS extensions. Quicksilver and Locus provide some support for transactions but have weaker guarantees. TxF and Valor provide file system transactions, while they argue for making everything a transaction. Paul Barham mentioned that Windows provides many types of transactions, but people still have a poor understanding of them.

Don then showcased their system. They extended the Linux 2.6.22 kernel to support transactions. They term it TxOS. It is based on the lazy version-management technique to roll back failed or incomplete transactions. All transactions operate on their own copy of the data and commit the data when the transaction is done. For the specific example given above, the system would lock the file, make a copy of it, and then unlock it. This is made still more efficient by using copy-on-write and other techniques. Since the technique does not hold any kernel locks, there are no risks of deadlocking and the operations always happen on private copies; when committing the transaction, the file is relocked, the changes are propagated, and then the file is unlocked.

The implementation of the system added 8.6 klocs to the system and required modifications to 14 klocs, with the goal of simple use. Among the performance measurements, there was a 40% increase in a dpkg install.

David Mazières said that he does not use such system calls but uses sockets and the NFS interface to access files, to which Margo replied that certain techniques work but this is a general mechanism. Michael Scott said that their use of lazy concurrency control may not always work, since not all things can be modeled as such, for example, I/O. The question was which parts of the system can they support and which can they not. Donald replied that they are not sure which parts of the system they can currently support.

- ***Your computer is already a distributed system. Why isn't your OS?***
  Andrew Baumann, Simon Peter, Adrian Schüpbach, Akhilesh Singhania, and Timothy Roscoe, ETH Zurich; Paul Barham and Rebecca Isaacs, Microsoft Research, Cambridge

Andrew described how modern multicore architecture increasingly resembles a network, so operating systems should be designed as a distributed system, not as a multi-threaded program. He showed a figure of an eight-socket machine with four AMD cores per socket. The picture looks very much like a network, with interconnect latencies varying from core to core and a fairly complex interconnect with a routing table. It will be difficult to design a shared data

structure to work efficiently on such a complex system and even harder to make it portable on different types of machines. Also, systems increasingly have many heterogeneous components, such as programmable NICs and GPUs. Then there are dynamic changes such as hotplugable memory, cores that can fail, and general power management. All these observations point to the machine exhibiting properties of a distributed system, so it should be treated as one.

Andrew showed the implications of treating the machine in such a way by a simple example comparing the costs of message passing and shared memory access. Accessing remote cache is like performing a blocked RPC, with cores blocked waiting for the cache lines to arrive and the operations limited by the latency of the interconnect round trips. This can be optimized by instead using nonblocking RPC such as sending a message to the remote server to perform the modifications. Messages are better because it is easier to reason about them, it decouples the system structure from the inter-core communication, it supports heterogeneous nodes, and it can even work without cache coherency.

Andrew discussed the trade-off of message passing vs. shared memory. Messages can be more expensive when the amount of data to be modified is fairly small. When using messages, state has to be replicated and partitioned between cores. Such techniques were already used in Tornado, K42, and clustered objects. This changes the traditional programming model: instead of blocking on operations, operations are split-phased, which ends up being a trade-off between latency and overhead. This also helps with heterogeneous architectures, since only the communication between different cores needs to be supported, and other parts of the system can be core-specific.

Andrew introduced the multi-kernel architecture, where, instead of one giant kernel, each core runs an individual kernel. This does not constrain the applications; they can still use shared memory over as many cores as they desire. Andrew suggested some optimizations to this design. Sometimes the message-passing default can be too heavyweight, such as for tightly coupled cores; in such cases shared memory should be supported.

George Candea suggested that this technique could be used to provide reliability as well, with resources granted by using leases. Could Andrew provide any insights into using something similar? Little is known about how to deal with hardware failures, but this technique can be employed to cope with software failures. Leases can also help in figuring out how much optimization is required for message passing. Steve Hand asked what kinds of services and applications will work on this system. They have studied a few core applications such as image processing, and other types designed for manycore workloads. They also want to support running many general-purpose applications and ensure that the OS does not get in the way of scalability. What happens if you instead run a VM on each core? It may well turn out that this architecture will end up looking quite similar to the proposed multi-kernel architecture.

HARDWARE

No reports were provided for this session.

THINK BIG

This was a discussion session.

Summarized by Vitaly Chipounov (vitaly.chipounov@epfl.ch) and Cristian Zamfir (cristian.zamfir@epfl.ch)

- ***Teaching Concurrency***
  Michael Scott, University of Rochester

Michael asserted that the current way of teaching concurrency is broken: "we are setting out to teach undergraduates what we have not yet, despite forty years of effort, figured out how to do ourselves, namely how to write parallel programs." Usually, people teach concurrency in an OS course by starting with Peterson's algorithm and then introducing locks, semaphores, etc. However, Michael complained that this approach to teaching is low on motivation.

Michael advocates introducing concurrency at every level of the curriculum, following a top-down approach, instead of teaching it solely in the OS course. For example, it is possible to talk about it in Web programming or programming languages courses. Message-based concurrency could be taught in networking courses. To avoid the need to teach intricacies like data-race freedom or memory models right from the start, he proposed encapsulating all these functionalities in high-level libraries and using them as needed.

Michael argued that there is a need for a language with built-in concurrency. He compared the concurrency in Algol 68, Java, and C#: while Algol can need as little as two lines of code to execute two statements in parallel, Java would need a page of code. C# would need slightly more than Algol. This is why he proposed C# as an alternative for teaching concurrency.

Timothy Roscoe argued that some people fiercely oppose this kind of approach, because people stop half-way and then specialize without understanding the low-level components. In many cases they do not understand hash tables or linked lists. In the worst case all they know is how to put together lines of code in an IDE. Michael replied that he was not convinced that somebody who just wanted to become a professional programmer needed to understand the memory model. If they understand data-race freedom, that's probably enough. David Andersen thought that it is better to teach students distributed operating systems first, and if they are really interested in the lower-level details, they should take an OS course.

Margo Seltzer argued that young students who learn to program Lego robots are already familiar with a language that expresses concurrency. This language is visual and the

students explicitly see the parallelism. She argued, however, that the academics are trying to unteach that when the students enter university.

- ***QoI >> QoS***
  Kimberly Keeton, Hewlett-Packard Labs, and John Wilkes, Google

Kimberly Keeton and John Wilkes explained why the quality of information (QoI) is more important than quality of service (QoS). They argued that what is done with data is probably much more important than whether the system is fast. They also presented metrics for information quality (IQ). Most of the talk consisted of real-world examples emphasizing the importance of quality of information. For instance, they recalled the NATO bombing of the Chinese embassy in Belgrade in 1999 because the data that led to that decision was inadequate. Another case for IQ is a sensor network monitoring earthquakes. Poor IQ could, for example, lead to a bad decision about whether to shut down a nuclear power plant, leading to severe financial consequences.

Some of the presented metrics for IQ included the freshness of the measurements and the level of aggregation (too much aggregation could lead to the eviction of outliers, potentially masking problems). Metrics can be discrete (reliable/not reliable) or continuous (e.g., relevance of a search result). Finally, metrics can be either context independent ("stand-alone") or context dependent. Kimberly argued that the stand-alone and context-dependent metrics are not the same and the role of research is to understand what is appropriate to measure.

The speakers also argued for tracking the IQ as information is flowing through the system, including cross-correlating data from multiple sources. They pointed out trade-offs between IQ and metrics such as performance, energy, or reliability. Margo Seltzer remarked that collecting provenance transparently is hard. John replied that low-hanging fruit might be attainable (e.g., error bars for the graphs in papers). Finally, the speakers indicated that database people have been researching IQ for a long time and we also needed to understand it in the context of systems.

- ***Sustainability***
  Geoffrey Werner Challen

Geoffrey explored the problem of sustainability in the IT industry. He presented different aspects, such as energy consumption, efficiency, obsolescence of equipment, and recycling. He drew an analogy between computers and cars and noted that, despite technological advances, the average number of miles per gallon had remained constant over the years. According to him, the main reason for this is acceleration: today's cars have the acceleration equivalent of the sports cars of the seventies. He then wonders whether our desktop computers are equivalent to 2008's Hummers.

The audience talked about ways to reduce the energetic footprint of IT. Armando Fox argued that it would be better to run computer-intensive experiments in the cloud, e.g., on Amazon EC2, instead of investing in dedicated clusters. George Candea proposed discouraging universities from buying new equipment. He argued that EPFL should introduce a new line in the IT budget, "IT services," which could be used to purchase EC2 credits.

They then discussed the problem of idle desktop computers that are never turned off. An audience survey showed that most of the attendees did not turn off their desktops for the duration of the conference. One participant remarked that computer systems are often left on because they need occasional network presence. He referred to two papers at NSDI '09 that proposed powering off the computer while using the network card as a proxy to do things like BitTorrent.

Michael Scott brought up the issue of obsolescence of equipment. He argued that, in the US, people discard 100 million cell phones per year, although many of them are still functional. He asked whether we could make use of this hardware instead. Margo Seltzer remarked that recycling is often done in Third World countries without concern for environmental safety.

Finally, Armando Fox remarked that in universities electricity is not directly billed to the users. Thus, people will probably not realize the importance of sustainability until there is a clear incentive, whether financial or political.

- ***Email Is Dead***
  Armando Fox

Armando argued that most people prefer instant-messaging (IM) and social networks to email. Email is still used for formal communication, but certainly for informal communication it is deprecated. Moreover, 90% of the email traversing long-haul networks is spam. There is also a certain cost associated with fighting spam, starting from the cost of filtering, the extra hardware resources, and the effort of people innovating in that area.

In a dialog with Margo Seltzer, Armando argued that whitelisting is not scalable, and he cited faulty email delivery between the two of them. However, social networks have the property that messages can only be sent to friends. The fundamental question raised is if there is any functionality of email that cannot be replaced with a combination of IM and social networks.

An audience member argued that email is fairly decentralized and it would not be scalable to have everyone subscribed to the same trust management system. Armando replied by asking what the distribution of email providers is and if it is not already the case that most people host their email at a few major sites (e.g., Gmail).

Timothy Roscoe argued that social networks are also exposed to spam and Colin Dixon said it is unrealistic to assume that everyone keeps their Facebook password safe. Armando stood his ground, maintaining that the term "social network spam" is underdefined at the moment. John Wilkes gave an example of spam on Facebook: people who inform

him of "every move in their universe," which is spam, vs. people who send him messages for professional reasons.

Dejan Kostić argued that email has the very important feature of plausible deniability. Another speaker said that searching IM logs is hard; usually communities who discuss an issue on IRC will later summarize the discussion in an email. Armando countered by saying that often people also summarize long email threads and that normally we do not use our email as a primary repository of useful knowledge.

John Wilkes and David Andersen thought that the main limitation of all means of social communication is lack of good access control management: that is the problem to be solved first.

DON'T TOUCH THAT DIAL

Summarized by Akhilesh Singhania (akhi@inf.ethz.ch)

- ***Security Impact Ratings Considered Harmful***
  Jeff Arnold, Tim Abbott, Waseem Daher, Gregory Price, Nelson Elhage, Geoffrey Thomas, and Anders Kaseorg, Massachusetts Institute of Technology

Jeff described the current practice of patching in Linux distributions. When an OS developer discovers and patches a bug, the patch is assigned an impact rating which the maintainers can use to prioritize which patches to apply. The problem is that assigning a bug a low-impact rating means it may not be patched right away, and detailed documentation of the bug gives hackers an easy tool to attack these unpatched systems. Impact ratings can thus actually be harmful to system maintenance.

Jeff gave the example of the sudo bug from 2001, which allowed an attacker to control a pointer used by syslogd. This was given a low impact rating, but eventually the vulnerability was exploited by attackers. Similarly, in 2003, when a patch for a bug had been available for around eight weeks, many systems still were not patched and were compromised. A member of the audience suggested that only two attacks in 15 years is not a bad track record. Jeff pointed out, with the help of a figure, how many bugs were disclosed but not rated and the number of days it took to give them a CVE rating. There is a fair delay between when a bug is found and when the security impact for it is assigned.

Jeff said that OS vendors and maintainers should not distinguish between security updates and other bug fixes and should apply them in a timely manner. Applying patches frequently is problematic because the system or the software often needs to be restarted. Therefore, they suggest using the hot update techniques (called Ksplice) laid out in their previous work to avoid the hassle of restarting the system.

Someone questioned whether people really care about keeping the system up-to-date. They still use older versions. Does the argument work for typical applications? Jeff replied that they are trying to address the core of the system and are not sure about what happens for applications.

- ***If It Ain't Broke, Don't Fix It: Challenges and New Directions for Inferring the Impact of Software Patches***
  Jon Oberheide, Evan Cooke, and Farnam Jahanian, University of Michigan, Ann Arbor

Jon showed statistics of recent Linux kernel vulnerabilities, taken off data from http://www.milw0rm.com, revealing the continued vulnerability of software, with security alerts coming out frequently. To address this, they have developed PatchAdvisor to automatically infer the impact of a patch on a software system so that system administrators won't have to assess the impact of a given patch on the data center.

Applying all available patches all the time quickly exhausts the resources of system administrators and may also have adverse effects on the patched system; patches sometimes introduce new bugs, cause incompatibilities and regressions, or might have other unintended negative impact on the reliability, performance, and security of software. A survey on the number of patches of production issued on Gentoo systems shows that a system administrator would need to review and deploy one patch per hour to keep up with the issue rate.

Matt Welsh wondered whether a lot of the presented patches for Gentoo are for programs that are never run. Why not just patch a program when the user first runs it, instead of all the time? Jon agreed and said their work actually went along those lines. A discussion about whether system administrator burden is a problem ensued, based on different views of the dimensions of the data centers that a system administrator has to patch.

Jon explained that the basis for PatchAdvisor is to patch common code paths as a middle ground between the two extremes of not patching at all and always patching, as these have a greater (positive and negative) impact on the total functionality of the system. PatchAdvisor is able to infer this impact via a combination of trace and static analysis to determine code coverage. Finally, he presented a preliminary evaluation of a patch to the psycopg2 package, which forms part of a bigger Web application suite. He argued that Web application suites provide a good evaluation opportunity because they exercise many layers of operating system and application code.

Future directions for the work are to improve the current ranking heuristics, to see if bugs cause great impact even in seldom executed code portions, whether application-specific knowledge about a bug or patch can be incorporated into the tool, and whether composite patches can be sliced into individual bits, removing areas of high risk. Also, the problem of classifying a patch to its purpose (bugfix, performance upgrade, security patch, etc.) might be addressed by their group.

Michael Scott said, Suppose your tool tells me that there is a lot of overlap between the patch and the code I run. What exactly is this supposed to tell me? Jon answered, To test better and be careful. Scott then pointed out that PatchAd-

visor is telling me that this patch is likely to be something I really need, while at the same time it might be very dangerous to apply it. Jon said that's the eternal question. The important and difficult part of this work is to find out what the trade-off to applying a patch is.

OUTRAGEOUS OPINIONS, OPEN MIC, AND HAPPY HOUR

Summarized by Vitaly Chipounov (vitaly.chipounov@epfl.ch) and Cristian Zamfir (cristian.zamfir@epfl.ch)

Dan Wallach made two points. First, vast available hardware resources are virtually unused. Even though the community is driven by performance, we should consider more algorithms and systems that can make use of these resources even though they are more complex (e.g., O(n^3) algorithms, as long as n is reasonably small).

The second point was that the conference reviewing/submission system is broken and there are a lot of papers that get resubmitted to many conferences even though they do not seem to have a chance. Dejan Kostić argued that those papers are not a problem and that the difficult ones are the ones in the middle. Dan proposed to borrow the model from the cryptography community: once a paper has been submitted, it is immediately made public as a technical report. He suggested that since USENIX is quite flexible and more willing to embrace new ideas, it can lead the way in improving the citation/tenure/review system.

Michael Scott mentioned the battle for making conferences more important than journals in the systems community while the main journal, *Transactions in Computer Systems* (TOCS) is losing importance. Matt Welsh said that the turnaround time for TOCS is extremely high.

Prabal Dutta suggested using an FAQ per paper that ACM should keep as part of ongoing dialogs. Margo Seltzer continued to discuss the concept of a "living paper," and Matt Welsh and David Mazières argued for, respectively, a blog/wiki and a forum model to represent the content. David also suggested that such an open space for discussion will prove useful for reading groups.

Steve Hand proposed to do something similar for the History of Programming Language Conference (HOPL) for the systems community.

Matt Welsh also proposed that we archive videos of the talks and at least convince speakers to provide the slides. Ellie Young replied that this is already done for most USENIX conferences.

Several members of the audience discussed making reviews public. Timothy Roscoe argued that for SOSP, reviewers can opt for making the reviews public. Margo Seltzer expressed her concern that these reviews do not represent the final version of the paper.

On a related note, Matt Welsh and Steve Hand commented on anonymity of the reviews and an analogy to the judicial system, where judges publish their opinions in the public records and are not allowed to maintain their anonymity.

George Candea gave an example of how short rebuttals can change the PC decision about a paper. The audience also discussed how PC meetings can make reviewers change their reviews, which makes the review process look biased. Finally, everyone pleaded for reproducible results, which makes papers more convincing.

GETTING A BETTER HANDLE ON DISTRIBUTED SYSTEMS

Summarized by Qin Yin (qyin@inf.ethz.ch)

- ***Simplifying Distributed System Development***
  Maysam Yabandeh, Nedeljko Vasić, Dejan Kostić, and Viktor Kuncak, EPFL

Maysam talked about how to make choices at runtime to gain better performance. The current practice of inserting a choice-making strategy into the basic functionality of distributed systems leads to complexity and more bugs. He proposed a new programming model for distributed systems: the application explicitly exposes to the runtime the choices it needs to make and the objectives it needs to achieve, and with the aid of a predictive model, the runtime support will make the right decision based on the current status of the environment.

One way to express choices is to implement a distributed system as a state machine with multiple simple and applicable handlers, which have simpler code and thus fewer bugs. Developers need to expose high-level objectives of safety, liveness, and performance for the runtime support to maximize. One possible implementation of the runtime is the predictive model inspired by Maysam's previous work, CrystalBall. The predictive model considers every choice and the consequences of the applicable handler, and resolves the choice by state-space exploration for performance.

John Wilkes mentioned relevant work from the International Conference of Autonomic Computing (ICAC), and Matt Welsh commented that a related field is control theory, which is used for tuning dynamic systems. Maysam said that the choice in his work is not resolved at development time but left to a sophisticated runtime system. Matt asked whether pushing the complexity to the controller will create fewer bugs. Maysam answered that the separation makes the main function simpler, and the common knowledge in the library can be shared by different modules.

- ***Automated Experiment-Driven Management of (Database) Systems***
  Shivnath Babu, Nedyalko Borisov, Songyun Duan, Herodotos Herodotou, and Vamsidhar Thummala, Duke University

Vamsidhar argued that in current systems, management techniques are limited and inadequate for end-to-end system management. Vamsidhar showed the importance of experiments in system management, introducing the con-

cept of experiment-driven management and the necessity of automating it.

Through a case study of an advisor for tuning database configuration parameters, Vamsidhar dissected experiment-driven management and talked about how to set up experiments, where and when to run experiments, and which experiments to run. Representative workload and data are necessary to set up experiments, which can only use underutilized resources in the production environment and never harm the production workload. Due to cost and time limitations, good algorithms to find the best subset of experiments are also important. In the case study, Vamsidhar proposed an experiment-selection algorithm called "adaptive sampling," which starts with a small bootstrap set of experiments and then conducts experiments based on estimated benefits and costs. He concluded that experiments should be supported as first-class citizens in database and general systems, with the cloud providing the foundation for a powerful workbench for automated, online experiments.

John Wilkes recommended research work in Duke about measuring and building models of NFS. Matt Welsh asked whether production systems have already done some work for online model construction. People thought that companies do performance experiments on their production systems to tune online provisioning.

- ***FLUXO: A Simple Service Compiler***
  Emre Kıcıman, Benjamin Livshits, and Madanlal Musuvathi, Microsoft Research

Large-scale Internet service is difficult to architect because of performance, reliability, and scalability requirements, but these requirements exhibit common architectural patterns, such as tiering, partitioning, replication, data duplication and de-normalization, and batching long-running jobs. Emre pointed out that these patterns have been redesigned and reimplemented according to measurable metrics such as component performance, resource requirements, workload distribution, persistent data distribution, read/write rates, and intermediate data size.

Emre introduced FLUXO, whose goal is to separate an Internet service's logical functionality from the architectural choices. Using a simplified social news service as an example, Emre explained how FLUXO maps high-level description down into an implementation with caching, replication, and service partitioning performed automatically. FLUXO works by accepting dataflow programs with annotations (such as consistency requirements and side-effects), keeping detailed runtime tracing, analyzing runtime behavior, performing programs transformations in the performance optimization space, and outputting a deployable optimized program.

Matt Welsh asked whether the developers will have to dig down into the generated programs to understand the mapping from high level to low level. Emre admitted that it's possible that developers will dig into the generated code to find bugs in FLUXO or do extra tweaks for performance improvement. Steven Hand asked how practical the extracted architectural patterns are. Emre replied that they investigated high-level diagrams of several Microsoft internal services as test cases and discovered that most services are logically simple and mostly use hash tables. Colin Dixon asked how to show that FLUXO is a better idea than the handout systems. Emre pointed out two important benefits: agility, and more efficient resource use. Timothy Roscoe asked about the relationship between FLUXO and Web service choreography. Emre's opinion was that Web service choreography is involved more with semantic issues of logical functionality integration than with system performance availability problems. Jeff Mogul commented that the problem is not only that of optimizing on a fixed infrastructure but also adjusting to workload changes and making decisions on the right infrastructure scale.

LEVERAGING EMERGING TECHNOLOGY TRENDS

Summarized by Adrian Schüpbach (scadrian@inf.ethz.ch)

- ***Reinventing Scheduling for Multicore Systems***
  Silas Boyd-Wickizer, Robert Morris, and M. Frans Kaashoek, Massachusetts Institute of Technology

Silas argued that caches on current multicores are underutilized. He proposed a new type of scheduler to overcome this problem. Caches are crucial for the performance, because access to main memory is slow. He said that an application with many threads and a big working set should fill first the L1 caches, then L2 caches and L3 caches, and go to main memory only when they are full.

He proposes a scheduler that focuses on data affinity, fits it to caches, and decides where to run threads. They implemented a prototype, called O^2. It assigns objects to caches and migrates threads to objects. Threads are also loaded to the cache of the same core. If a thread starts manipulating another object, load it to another core's L1 cache and migrate the thread to that core. Then migrate the thread back to the original core so that the thread can continue to manipulate the original object.

Silas identified the two operations: o2_start(id), which marks the start of an operation and is also the point where a thread might migrate to another core, and o2_end, which marks the end of an operation and is also the point where a thread might migrate back to its original core.

Someone wondered if the assignment of data to caches can be complex. Can the overhead not be quite large? Sure, it can, said Silas. Might it be that threads migrate all the time? Silas wondered why that is a problem. Someone else said it is not always the case that the threads go where data is. Threads need to access objects, but also parameters to methods and globals. Is it cheaper to move threads to objects or might it be cheaper to move objects to threads where parameters are? They use statistic counters to find out whether to move threads or objects according to cache misses. Someone pointed out that since parameters to

methods should hopefully be in the shared L3 cache, they are accessible from all the cores. Can you control that by explicit cache instructions? Silas replied that this would be interesting to look at.

- ***FAWNdamentally Power-efficient Clusters***
  Vijay Vasudevan, Jason Franklin, David Andersen, Amar Phanishayee, and Lawrence Tan, Carnegie Mellon University; Michael Kaminsky, Intel Research, Pittsburgh; Iulian Moraru, Carnegie Mellon University

Vijay pointed out that power has become an important issue in the last few years and that it always was an issue in chip production. Now it is very important in data centers. Google places data centers according to the power infrastructure. The goal is to increase the efficiency of the infrastructure of data centers by using dynamic power scaling.

FAWN (fast array of wimpy nodes) consists of an array of well-balanced low-power systems and reduces the amount of energy to do data-intensive computing. The prototype is built with a 4W AMD Geode with 256MB DRAM and a 4GB compact flash card. Vijay claims that whole data centers can be built using these nodes.

Vijay provided four reasons why FAWN should be used. First, fixed power costs dominate and using DVFS only does not minimize the power consumption of a whole node, since CPUs don't dominate power consumption. Second, FAWN balances energy consumption: in traditional approaches the CPU-to-disk ratio grows, and a CPU needs power even if it is waiting. Third, it targets the "sweet spot in efficiency." The fastest CPUs are inefficient in that they need too much energy per instruction, because they need transistors for speculation and out-of-order execution. Finally, FAWN reduces peak power consumption, which is important for cooling, power supplies, and UPS. Vijay showed some energy-per-instruction results.

Someone asked what the lifetime of FAWN is, compared to traditional systems. Vijay replied that it is used in embedded systems and it lives long. Roscoe pointed out that more nodes also means more networking. Did they consider the costs of cooling networking gears and switches? They don't necessarily need more networking and haven't considered these costs yet. John asked if the performance measurements are throughput-based, not latency-based, and Vijay responded affirmatively. John pointed out that we also have latency, not only throughput, and that might give more bounds not shown on Vijay's graph. Vijay agreed. Why haven't Google data centers, for example, not yet moved to low-power machines? Vijay didn't know, but it could be because they invested a lot in traditional systems and cooling systems.

WRAP-UP TALK

Armando Fox, Program Chair

Summarized by Tudor Salomie (tsalomie@inf.ethz.ch)

Armando revisited some of the topics that he considered the most interesting:

1. From Adam Greenfield's talk about networked urbanism: we should follow the effects of going from passive to networked resources to their social and logical conclusions. We should switch from passive objects to network services.

2. On the topic of sustainability, we need to look into funding models, what we should do when talking to people who dispense money, and how we should avoid having idle machines.

3. Regarding the conference submission process, the idea of having living papers and of a dialog beyond the review process should be considered. Maybe we should also rethink the role of a journal and that of a conference, as it was pointed out by Michael Scott: we got what we asked for, but is that what we really wanted?

4. Teaching concurrency is important. Is the way we teach concurrency for distributed systems the same way we should be teaching it for multicore systems?