HotOS X: Tenth Workshop on Hot Topics in Operating Systems

Sante Fe, New Mexico
June 12–15, 2005

RELIGIOUS WARS

Summarized by Alexandra Fedorova

- ***Are Virtual Machine Monitors Microkernels Done Right?***

  Steven Hand, Andrew Warfield, Keir Fraser, Evangelos Kotsovinos, and Dan Magenheimer, University of Cambridge Computer Laboratory and HP Labs

  Steven Hard argued that microkernels and virtual machine monitors (VMMs) both emerged to achieve isolation of the software system from the underlying hardware, but used different means to do so. He highlighted similarities and differences among VMMs and microkernels, talked about architectural lessons learned with these systems, and suggested that it is best to design architectures that borrow the best from VMMs and microkernels, as opposed to sticking to a particular architecture.

  He summarized the differences as follows: Whereas VMMs multiplex entire operating systems, microkernels multiplex many small tasks as threads. VMMs are closely aligned with hardware, so the interface looks like hardware; microkernels expose a higher-level interface, and tasks communicate using synchronous IPC. VMMs offer one address space per scheduled entity; microkernels offer multiple scheduled entities per address space. Architectural lessons learned from research with these systems are:

  - Avoid liability inversion. Moving trusted system code to the user level, as is done in microkernels, involves having to trust user code to perform essential system functions (e.g., user-level page in Mach).
  - Make synchronous IPC irrelevant. Synchronous IPC in microkernels is expensive. But as we learned from Xen, IPC does not need to be on a critical path.
  - OS is a reusable component. VMMs have achieved complete component reusability, because they treat OS as the reusable component.

  During the Q&A session, the most fire came from Gernot Heiser, who disagreed with the analysis. Gernot argued that liability inversion is not inherent to microkernels (in UNIX you have system daemons implemented as user processes that run in privileged mode). He also insisted that IPC in microkernels is not a problem: New fast hardware allows cheap implementation. Besides, Liedtke, using L4 as an example, demonstrated that microkernels can be fast.

- ***OS Verification—Now!***

  Harvey Tuch, Gerwin Klein, and Gernot Heiser, National ICT Australia

  Harvey Tuch argued that there is an urgent need to develop practical formal verification tools that can be used in high-performance industrial operating systems. This is a challenging task, and requires the right OS architecture: basically, a microkernel. VMMs won't work because they increase the size of the TCB.

  The rest of the talk was a survey of available formal verification methods. Formal verification is done by constructing a system model, a system specification, and a verification tool that checks for desired properties using the model and specification as input. Examples of formal specifications are HOL temporal and Bayer-Moore logic, microkernel APIs. Model checking is usually done by automatic reachable state-space exploration or by theorem proving. They have already implemented some preliminary verification tools for the L4 microkernel.

  During the Q&A, Aaron Brown pointed out that real customers are using OSes as giant application

servers; they run databases and application servers on them, and it is not clear how much leverage you are going to get from verifying just the OS kernel. Several other attendees, including Eddie Kohler and Rik Farrow, asked how they were going to deal with changing operating systems. Harvey responded that there are two types of changes, implementation changes and API changes; they have a way of dealing with implementation changes, but API changes are more difficult.

- ***Making Events Less Slippery with eel***

  Ryan Cunningham and Eddie Kohler, University of California, Los Angeles

  This talk continued the old debate concerning threads and events. Events are fast but difficult to program. The speaker described eel, a programming tool designed to simplify programming with events. eel uses program analysis techniques to improve programmability while preserving the event model, and provides the libeel event library, visualization, and debugging tools.

  Programming events is hard because it is difficult to understand the flow of the program and difficult to debug. eel's visualization tools make it easy to visualize the control flow, and the debugger allows you to step through each program flow separately; you follow the callbacks related to the same connection, so each flow of related events appears sequential. The speaker concluded that events don't need to be hard to program. You can use simple tools to extract program control flow and to help with visualization, verification, and debugging.

  Margo Seltzer pointed out that while eel helps clarify the event-based program after it's written, it does not appear to help with the actual writing of the program. The speaker responded that by allowing the programmer to visualize newly written code, eel does make writing easier. Margo was not convinced that using events is worth all this trouble. The speaker said that using events is preferable because they are much faster. Pei Cao responded that based on her experience of designing large software projects people use both threads and events. Events are a pain, so people use them only when performance is absolutely critical, which does not happen very often.

STORAGE

Summarized by Steve VanDeBogart

- ***Parallax: Managing Storage for a Million Machines***

  Andrew Warfield, Russ Ross, Keir Fraser, Christian Limpach, and Steven Hand, University of Cambridge Computer Laboratory

  Andrew Warfield spoke about Parallax, a system for dealing with the increased demand on storage systems created by the use of virtual machines. With the adoption of virtual machines in cluster environments, the number of system images per disk has increased by a factor of 10 to 100. Some organizations expect that within the next few years they will be supporting one million live system images in a single data center. Furthermore, techniques that take advantage of historical versions of VM state (e.g., for intrusion detection or configuration debugging) imply additional demand on storage systems. Parallax is designed to solve these problems by being able to scale to many images while supporting fast and frequent snapshots.

  The key insight behind Parallax is that block-level write sharing should not be on the critical path. The common case is that all active system images started from a small number of base images and then diverged. While it is possible that distinct systems have the same software added on top of the base image, this is not the major source of shared blocks. Parallax handles this by maintaining a radix tree of the blocks in the active image.

  Armando Fox and Mary Baker asked about the fault tolerance of the system. Andrew responded that nodes are monitored for failure (both loss of connectivity and deadline failure) and that blocks are sent to at least three nodes to reduce the chance of losing information.

- ***Stupid File Systems Are Better***

  Lex Stein, Harvard University

  Lex Stein discussed some experiments he conducted to validate the file system speed tricks accumulated over the last 30 years. The classic assumption in file systems is that the closer the block numbers, the faster it will be to access blocks. However, with virtualized block numbers on modern storage systems, this is not necessarily the case. To determine how much the smart allocation of blocks affects performance on a modern storage system Lex removed the smartness by randomizing the block values in a trace.

  Two traces were used to conduct tests, one taken during the compilation of a Linux kernel, the other under the Postmark benchmark. The block numbers in these traces were then randomly permuted. The modified and unmodified traces were played back in a disk-accurate simulator with a varying number of disks used as JBOD (Just a Bunch of Disks), RAID 4, and RAID 5. The simulation results show that with a modest number of disks the randomized traces started to perform better than the smart traces.

  John DeTreville suggested that maybe the smart traces were trying to schedule the blocks into too short a time frame and thus not taking advantage of the parallel seek capacity of the multiple disks. Lex replied that maybe there is a point where things should get parallelized and that this might be a new trick to improve the perfor-

mance of file systems. Russ Cox added that maybe instead of dumbing down the file system, since it still does help performance on a single disk, the volume manager should do a better job of distributing blocks. Petros Maniatis suggested adding traces that evenly distributed the work among the disks for comparison: maybe performance is improved, not by randomness per se, but through hot-spots eliminated in the random trace.

- ***Aggressive Prefetching: An Idea Whose Time Has Come***

  Athanasios Papathanasiou and Michael Scott, University of Rochester

  Athanasios Papathanasiou put forth the idea that the performance characteristics of common memory and storage architectures have changed enough that in order to get good performance (where performance may affect energy consumption), aggressive prefetching must be done. There have been drastic improvements in CPU power and storage capacity with moderate improvements in I/O bandwidth, but I/O latency has hardly improved at all. This means there is decreased risk and increased need to do prefetching.

  Bandwidth has increased 40% per year, but latency has only increased 10%. In order to lower the latency to the same level as before, increased prefetching is required. Furthermore, the amount of memory available to do speculative operations has increased. Memory slack has increased by a factor of 100 over the past 15 years.

  Having pitched the idea of aggressive prefetching, Athanasios presented some research challenges that would have to be met in order to make aggressive prefetching worthwhile: device-aware prefetching; characterization of an application's I/O demand; coordinating I/O requests with device power states; speculative predictors that provide sufficient data coverage; and better metrics to account for the true cost of cache hits and misses.

  Several attendees pointed out that good access models are needed in order to prefetch the right data. Athanasios agreed, adding that we need to come up with some generic models with parameters that can be tuned for specific devices or applications.

OUTSIDE THE COMFORT ZONE

Summarized by Steve Zhang

- ***Why Markets Could (But Don't Currently) Solve Resource Allocation Problems in Systems***

  Jeffrey Shneidman, Chaki Ng, and David Parkes, Harvard University; Alvin AuYoung, Alex Snoeren, and Amin Vahdat, University of California, San Diego; Brent Chun, Intel Berkeley Research Lab

  Jeffrey Shneidman argued for using markets to solve the resource allocation problem in large-scale systems. Utilization data from systems like PlanetLab shows how demand exceeds supply, especially during peak times, and demonstrates the need for an allocation policy. Using markets would allow for an efficient policy where those who value the resources the most receive them.

  He outlined some problems that would face any allocation policy. First, it would have to be selected and supported by the users. In addition, it would be difficult to divide resources, since different resources in a system may be connected in subtle ways. Consumers would also need to predict needs well and accurately value their needs based on the currency chosen. Finally, the implementation of a market policy requires a method for expressing bids easily and efficiently. Despite these problems, and although using market economies have been studied before, he believes today's environ-ment (e.g., demand much higher than supply, a semi-cooperative user base, repeated large resource allocation, improved OS support for resource isolation) could make revisiting this old idea fruitful.

  Several people expressed doubt that demand outpacing supply is only a problem for free usage testbed systems. Others noted the difficulty in choosing a viable currency for such a market economy. Although history has shown that artificial currencies tend to work poorly, using real money is generally unpopular with user bases, since users would not start on an equal footing. Finally, real economies have problems that are generally addressed by government-controlled regulatory agencies, and an analogous solution would need to be found for a market-based allocation policy.

- ***Operating Systems Should Support Business Change***

  Jeff Mogul, HP Labs

  Jeff Mogul expounded on why building systems with business changes in mind (e.g., starting or changing services, meeting new regulations, mergers, and spinoffs) is extremely important for most enterprises. AT&T wireless's downfall, HP's SCM rollout, and Comair's crew-scheduling fiasco were mentioned as anecdotal evidence of the difficulties in adapting to business changes and the high costs of being unable to do so efficiently. Agile businesses tend to dominate their markets. However, most business systems are so complex that changing or replacing them is extremely costly, and with most IT departments' current focus on cost-cutting, change is never easy or quick.

  Jeff believes that research should focus on allowing application-level flexibility, which requires standardization at the lower layers at enterprise scales. OS-level flexibility, which has been the historical focus of many researchers (e.g., microkernels, extensible OS), actually undermines standardization. Al-

though formal verification may be unrealistic for years to come, OS conformance testing should be a first-class research topic. Also, capturing an accurate snapshot of an IT infrastructure at many levels and quantifying the value of IT (how much money a machine or a system is making) would be vital for management to gauge the costs and benefits of system changes accurately. Finally, with the prevalence of outsourcing, auditability of systems becomes critical for third-party auditors to verify that a customer's requirements are being met by the supplier.

Researching these areas is challenging, however, because enterprise applications tend not to be open source and may require millions of dollars and several person-years to install and configure. Studying controlled testbed systems in a lab environment is not a replacement for looking at real deployed systems. In addition, it is not clear how to measure the success of any method in addressing these concerns.

IT'S NOT AI, IT'S SYSTEMS

Summarized by Steve Zhang

- ***Designing Controllable Computer Systems***

  Christos Karamanolis, Magnus Karlsson, and Xiaoyun Zhu, HP Labs

  As a prelude to Christos Karamanolis's talk, Elizabeth Bradley of the University of Colorado, Boulder, gave a brief introduction to control theory. She talked about simple methods and tools that work for linear time-invariant systems. However, operating system problems tend to be nonlinear and time-varying, the solutions for which are usually ad hoc. Although it's possible to simplify many of these cases to work with a linear system-based approach, it's very important to be mindful of the assumptions that must be made for these solutions to be valid.

  Christos Karamanolis talked about his experience in applying control theory to systems management. Recent surveys have shown that 75%–80% of IT costs go into managing existing systems. A feedback-based control system would allow humans to be taken out of the loop and thus cut costs immensely. However, because computer systems change quite frequently, adaptive control is needed to dynamically estimate models to be used by the controller.

  The authors experimented with using an automated controller as a scheduler that handles different classes of clients, and attempted to have the system maintain a consistent throughput based on a service-level agreement. Some important lessons were learned: First, more recent controller actions must have higher impact on current measurements than earlier actions. Second, the action-measurement relationship must be close to linear; where that is not the case, different actions and/or measurements must be tried until a linear pair is found. More properties from control theory were translated into system requirements in order to help system architects design systems more amenable to automated control, and these are listed in the paper. During the Q&A session, it was established that although control theory is at least four decades old, researchers have only recently explored taking formal control-theory approaches to systems issues. However, informal ad hoc methods based on control-theory principles have been used for quite some time.

- ***Three Research Challenges at the Intersection of Machine Learning, Statistical Induction, and Systems***

  Moises Goldszmidt and Ira Cohen, HP Labs; Steve Zhang and Armando Fox, Stanford University

  Terran Lane from the University of New Mexico set the stage for this talk by providing an overview of machine learning, focusing on supervised learning and reinforcement learning. In supervised learning, the goal is to build a model that can predict system output from sensor values, whereas in reinforcement learning the goal is a model that can control the system according to sensor values to achieve some desired behavior. For supervised learning, there are well-established techniques that can handle high-dimensional data, but reinforcement learning techniques work best for low- (5–10) dimensional data. The speaker only briefly touched upon unsupervised learning but believes that many system problems may require such techniques.

  Moises Goldszmidt then talked about the challenges of applying any statistical learning technique to systems. More specifically, he related the problem to his work with correlating low-level system metrics with higher-level objectives. Finding such correlations not only would help novice system administrators deal with simple problems, but would also be useful to experts by providing better visibility about the behavior of large-scale systems.

  Although there is generally plenty of raw data available for problems in this arena, thanks to mature measurement and monitoring tools, there is a lack of labeled data that would aid the evaluation and comparison of different approaches. Another challenge is that learning schemes must be adapted to handle online streams of data. Although the machine-learning community has yet to provide any general solutions, it is possible to use domain-specific knowledge to construct efficient algorithms for the systems area. The speaker also noted that in most cases, obtaining true root-cause analysis is neither practical nor necessary. Instead, one should strive for diagnosis that can easily map to possible repair actions.

- ***Panel: Control Theory/Machine Learning***

  Christos Karamanolis, Elizabeth Bradley, Terran Lane, Moises Goldszmidt

  This panel focused on the feasibility of applying control theory and machine-learning techniques to large-scale distributed systems. The consensus seemed to be that without centralizing data from individual nodes, it would be very difficult if not impossible for these approaches to effect conformance with systemwide high-level policy. It was noted, however, that while a globally optimal solution may be impossible for distributed systems, locally stable methods (e.g., TCP/IP) are often practicable.

  This led to a reference to biological systems and how nature has solved the problem of achieving global goals from local control. Terran Lane responded that Mother Nature has reached such solutions through billions of years of experiments on an infinitely parallel supercomputer. He added that if one had a problem that had an accurate analogy in nature, using nature's solution would be feasible. However, most problems do not fit such a model.

CLEANING UP THE MESS WE'VE MADE

Summarized by Alexandra Fedorova

- ***Making System Configuration More Declarative***

  John DeTreville, Microsoft Research

  System configuration is hard. A huge fraction of every user's time is spent futzing in a computer system. This is the biggest performance bottleneck. The issue is that a configuration is a shared mutable state. We update the state in place when we install and uninstall. A system's correctness depends on every install and uninstall we have ever done. The proposal is to use declarative configuration: record everything that describes system configuration, all files, all system variables. Then we have a chance of checking whether the configuration is correct. But in order to do this, we need a system model.

  The system model can incorporate submodels. Programmers, publishers, and remote administrators can write these submodels. Models express rules for composing the programs into systems. The expectation is that system models will be easier to compose from submodels. As a result, no sequence of installs and uninstalls can result in a badly formed system instance. The drawback of this approach is that system models/policies may be too difficult to express. In conclusion, John pointed out that earlier efforts at declarative configuration were not widely adopted, because they were targeted at programmers.

  Joe Hellerstein asked how to deal with distributed applications. John said that this is a hard problem but that improving local system administration is a good start. Jay Lepreau pointed out that standard program installers are already designed to handle program interdependencies. John responded that they have the right mechanisms, but in practice they do not handle these interdependencies properly.

- ***Reducing the Cost of IT Operations—Is Automation Always the Answer?***

  Aaron Brown and Joseph Hellerstein, IBM T.J. Watson Research Center

  Aaron Brown said that costs of IT operations are quickly outpacing server spending. A common solution is automation, but expenses involved in developing and deploying an automated solution often outweigh the savings it provides. Aaron provided a case study where automated solution did not help to cut costs. Then he offered a mathematical framework for evaluating the cost-effectiveness of a solution.

  Basic cost model: There are fixed costs for setup and maintenance and variable costs for automated inner loop and per-instance tasks. Because there are fixed costs involved, you have to look at the lifetime of automation, to amortize fixed costs over time. Automation lifetime can be very short, because the software package or the installer might change, for example. Apart from cost, there are the issues of trust and adoption. Automation is a disruptive force for IT systems managers. Using an incremental transition path from manual to automatic may be the right approach.

  Margo Seltzer suggested that it may be difficult to know the lifetime of automation. Aaron agreed and said that they would like to be able to predict it. Christos Karamanolis wondered how they can quantify fixed costs for the model. Aaron responded that this is challenging and that user studies are needed to evaluate the costs of complication that come with automation. However, they do have a way of modeling such costs.

- ***Human-Aware Computer System Design***

  Ricardo Bianchini, Richard Martin, Kiran Nagaraju, Thu Nguyen, and Fabio Oliveira, Rutgers University

  Thu Nguyen began his talk by encouraging the community to consider the human as a first-class entity in computer system design. He argued that systems designers should use human-aware principles, such as:

  - making systems more robust to human mistakes;
  - understanding human actions and mistakes;
  - developing techniques and infrastructure to increase human understanding of systems to prevent, hide, and undo mistakes;
  - designing new metrics and benchmarks to measure system improvements.

  He then described their work on understanding operator actions and mistakes in configuring Internet services. The results of this study could be used to fix and prevent

user mistakes. Thu concluded by admitting that designing good human-factor benchmarks is hard and recruiting human study subjects with the necessary background is even harder.

Pei Cao suggested that the reason why networking guys do a lot more online testing is because Cisco routers are specifically designed for online testing. If people built better software, maybe we would not have such issues with installation complexity. If you build complex software, you have to have a user model in mind.

APPROACHES TO OS RESEARCH

Summarized by Prashanth Bungale

- ***Thirty Years Is Long Enough: Getting Beyond C***

  Eric Brewer, Jeremy Condit, Bill McCloskey, and Feng Zhou, University of California, Berkeley

  Bill McCloskey explained that the point of this talk was to get people to think of C as a bad habit and to stop using it. Safety and security have now become more important than any other factors. Low-level systems are still unsafe and insecure, and C is the main problem. Java has failed mainly because it is not expressive enough and because porting applications is expensive. The authors believe that it's possible to design a systems language that is safer and better than C, Java, or C#. Areas for improvement include memory management (GC is not good enough), concurrency (manual locking is too error-prone), data layout (the programmer should have bit-level control on data storage in memory), and API adherence (compiler checks should be automatic).

  Their proposal for a new language, Ivy, guarantees that the following classes of errors are eliminated: buffer overflows (via bounds checking), dangling pointers (via checked memory management policies), race conditions and deadlocks (via use of atomic sections instead of explicit locking), API violations (via type qualifiers), resource leaks (via computation stacks), and macro errors (via a safer and newer preprocessor). In conclusion, Bill pointed out the problem of having two conflicting goals: starting out with a safe, clean foundation, or keeping existing code relevant.

  An audience member asked about what linguistic features were novel in Ivy, and why. The speaker said that this was future work. Prashanth Bungale asked why, when previous attempts such as Cyclone failed for this reason, we should believe that Ivy won't end up involving enormous amounts of human intervention. Bill replied that it is a fundamental goal of Ivy to reduce manual intervention as much as possible, but it remains to be seen how much this can be done. Jay Lepreau said that Cyclone on TCP/IP (which had millions of lines of code) actually hadn't required very much manual intervention, and that perhaps what kept Cyclone from being widely adopted were things like resistance and inertia.

- ***Broad New OS Research: Challenges and Opportunities***

  Galen Hunt, James Larus, David Tarditi, and Ted Wobber, Microsoft Research

  Galen Hunt described his working definition of OS research as research into the base abstractions provided for computation and into the practical implementations of those abstractions.

  Singularity is a research project with the following hypothesis: Sound verification techniques can be combined with new OS abstractions to provide dependability, reliability, and security (though sometimes at the expense of performance). It has focused on the following:

  - Configuration and manageability: The OS knows a lot about the hardware but next to nothing about the applications. Where's plug 'n' play for software applications?
  - Safe system extension: Extensions add new value to applications from the user's perspective (e.g., Google toolbar) but are unsafe. If we look at the last few SOSP submissions, safe OS extension has been an active area of research. But what about safe application extension?
  - Multi-processor cores: We can expect up to 256-processor cores in the foreseeable future, and hence need better OS support.

  The Singularity architecture includes a VMM for abstract hardware (with the abstract instruction set being type-safe, memory-safe MSIL), closed processes (i.e., no shared memory, no dynamic code loading, no dynamic code-generation), and IPC channels with contract guarantees.

  One key feature of Singularity is the concept of software-isolated processes (SIPs), where each process has its own garbage collector and its own garbage collection domain, exclusive ownership of its address space, and no pointers outside its address space (guaranteed because of type-safety). Therefore, to exploit this situation for performance reasons, the entire system is run in ring 0. According to their IPC micro-performance results, Singularity is an order of magnitude faster than hardware-protected systems, because there is no hardware protection domain change, and an order of magnitude slower than an in-process procedure call, because the IPC involves a GC domain change.

  Gernot Heiser asked if they also had a new hardware model; otherwise, how would type-safety help device drivers? Galen responded that their hardware model already incorporates simple I/O ports, for example, but it is not yet sophisticated. Armando Fox was concerned that while the changes regarding the hardware protection domain (e.g., entire system run-

ning in ring 0) may only change the performance from being "really fast" to being "really, really fast," why get rid of the mechanical intellect just for this reason? His concern was mainly that historically, checking through software has been hard to get right. Jay Lepreau pointed out that Andrew Appel broke type-safety in secure chips. Galen responded that if you care a lot, you can always put the process in a higher ring; an option can be provided to say, "OK, I don't trust this code. Use a separate protection domain." Robert Grimm commented that using an abstract machine as an executable platform made it hard to predict . . . The speaker interrupted him and immediately responded that everything is *compiled* (no more JIT).

- ***patch (1) Considered Harmful***

  Marc Fiuczynski and David Walker, Princeton University; Robert Grimm, New York University; Yvonne Coady, University of Victoria

  Marc Fiuczynski said that the key lesson from his talk is that we need better tools to improve OS evolution. The open source model is used everywhere—embedded systems, servers, clusters, HPC—and fosters community development. Updates are performed through patches, where the changes can correspond to intraprocedural changes (modifications to the internal logic of a function), intermodule changes (changes to a function's signature or a data structure's field-makeup), or behavior changes (changes to the semantics of interfaces). Through examples of Linux kernel patches containing separate concerns, Marc showed how an extension could easily cover a hundred existing kernel files, even though it represents a logical unit expressing a single, cross-cutting concern. Current practice makes OS evolution hard and dirty: Since it is hard to understand implementation of a concern, composition of separate concerns is very hard, maintenance is generally hard (or very annoying), and new concerns bloat and dirty the mainline code base.

  Marc then presented C4, a toolkit to improve OS evolution. The problem with the existing approach of patch (1) is that it operates at the lexical level. By contrast, C4 functions at the semantic level so that we can build better tools to manage complexity and analyze interference semantically. Their approach is to use aspect-oriented software development (AOSD) techniques. C4 provides a semantic patch compiler through which one can express behavioral changes as semantic patches using aspects, which provide a language-supported methodology for integrating cross-cutting concerns with a program. The C4 toolkit consists of an unweaver and a weaver, which are analogous to diff and patch, respectively. The main difference is that the unweaver actually removes the code belonging to an aspect from the baseline code and the weaver puts it back in the right place, using knowledge of C's abstract syntax to avoid merge conflicts.

  Their current focus is the engineering effort needed to get the weaver and unweaver working. Future work enabled by C4 includes program analysis tools, identifying data structure changes, identifying memory safety (or lack thereof), and capturing programmer intentions via declarative frameworks.

  Steven Hand commented that a lot of the problem simply lies in really crappy open source code. Margo Seltzer asserted that they are still distributing patches and asked what they distribute exactly. Marc responded that they distribute C4 files. Chris Small asked how they are going to get people to use these tools. Marc replied that whether elegant or not, they reduce the pain for the user, and that is what is going to get people to use their tools. Kirk McKusick asked why not use CVS. Marc responded that even then one would have to deal with merge conflicts. Jay Lepreau commented that the problem is that Linux has a pope model—there's only one integrator.

- ***Panel: Do We Work Within Existing Frameworks or Start from Scratch?***

  Bill McCloskey, Galen Hunt, Marc Fiuczynski, Robert Grimm, Russ Cox, and Eric Brewer

  Chris Small: Mike Jones once said to me, "When you're on an exponential path, nothing you do now matters." So, why waste your time on Ivy? Why not build a better language instead?

  Eric Brewer: Do both. We're not going to get the language right either; but Ivy is extensible.

  Robert Grimm: Every year, the programming language community publishes work and more work on Java + delta. What if you want Java + delta + delta? Extensions are the answer.

  Margo Seltzer: Galen's talk seemed so far removed from helping the end user.

  Galen Hunt: Extensible applications would help the end user to a great extent.

  Andrew Hume: Saying "The language we're programming in is the problem" is a delusion.

  Galen Hunt: The language directly affects what we think.

  Andrew Hume: That's balls.

  Eric Brewer: I don't believe any of that.

  Phil Lewis: Education—what languages do you learn? Let's not teach people C! Ten, twenty years from now, the problems will go away.

  John DeTreville: This is not about languages. Back in the '70s, people wrote their own OS, compiler, etc. But now it's impossible to write your own OS.

  Michael Scott: Regarding the panel question, look at past examples. Two and a half models for success: (1) Exponential curve: Java, Perl, HTML, etc.: there wasn't a market

for it. (2) Migration path: C++ (from C), XHTML (from HTML), Opteron (30 years of x86). (2.5) In between: MESA, Smalltalk, VKernel, maybe Plan9. So which model should we be looking at now?

Galen Hunt: My answer is a definitive it depends. Will I ship Singularity as a product? No, we will learn ideas.

Pei Cao: Galen's presentation's problem and solution seemed totally far off.

Galen Hunt: Then you young people should go work on solving the interesting problems.

Margo Seltzer: What lessons can the OS community learn from each of your projects, assuming that you're wildly successful?

Robert Grimm: Languages are very important for reliability and security.

Galen Hunt: Sound static analysis and verification can dramatically impact our ability to test (9,000 testers testing Windows currently, and you know the result).

Eric Brewer: We do want to support legacy drivers, etc.

Galen Hunt: The OS knows nothing about the application. We still have a 1970 model of what a program is: a.out, stdin, stdout, stderr model.

Andrew Hume: Sometimes you just get it right! Clarity and economy of expression . . . Are we ever going to have "the model" of an OS, or are we going to continually have periodic purging?

Galen Hunt: I don't know!

DISTRIBUTION

Summarized by Nikolaos Michalakis

- ***WiDS: An Integrated Toolkit for Distributed System Development***

  Shiding Lin, Aimin Pan, and Zheng Zhang, Microsoft Research Asia; Rui Guo, Beijing University of Aeronautics and Astronautics; Zhenyu Guo, Tsinghua University

  Zheng Zhang started by explaining that today's distributed system development process is unscalable for humans. Debugging is painful, and there is code divergence between simulation and implementation. WiDS is designed to maintain one code for both simulation and implementation, simplify debugging by allowing debugging in a single address space as much as possible, and support large-scale performance studies of the system in design. To achieve these goals, WiDS essentially lets programmers link their code to different libraries according to their needs (single-node simulation, parallel simulation, network execution). Verification and debugging of protocol implementations can be done through a model checker in simulation.

  Experience with WiDS shows that distributed system development and deployment can be greatly simplified both in building complete systems such as BitVault, a data retention system, and in the large scale, such as the RNRP protocol on two million nodes. Research in progress hopes to include playback of message logs in simulation mode to find network-related bugs. In terms of extending APIs, whether to use events or threads must be a programmability-centric decision. Zheng's conclusion was that distributed system and tool development should go together.

  Ion Stoica noted that in reality there are problems due to connectivity asymmetries, congestion, and unbounded packet delivery, and asked how many of those violations WiDS included in their simulation. Zheng replied that they have end-to-end connectivity simulation and packet drops and that the best way to incorporate newly discovered violations is to use the model checker and update the protocol model. Andrew Hume asked if Zheng had found anything not covered by the protocol checker. He replied that this could happen if, for example, the protocol specification was implemented incorrectly or if the model was wrong.

- ***Causeway: Operating System Support for Controlling and Analyzing the Execution of Distributed Programs***

  Anupam Chanda, Khaled Elmeleegy, and Alan Cox, Rice University; Willy Zwaenepoel, School of Computer and Communication Sciences, EPFL

  Anupam Chanda began by sketching the execution flow of a multi-tier program composed of a Web and database server. Execution steps are performed by "actors," such as system calls, over "channels," such as sockets. As he noted, it is sometimes useful to write meta-applications to control and analyze the execution flow of such multi-tier programs. Meta-applications can be categorized as "log-based" (e.g., Magpie) or "metadata passing" (e.g., Pinpoint). Unlike log-based approaches, metadata passing across actors allows online control of multi-tier programs and is the approach chosen by Causeway, a framework that provides OS support for building meta-applications.

  The framework is placed at the level of the OS, since placing it at either the application or middleware level might lead some components of the multi-tier program to be oblivious to metadata passing, might require modifications to all applications, and, in the case of middleware, might not support all legacy protocols. Causeway associates metadata with an actor upon a write on a channel and propagates

metadata when the actor at the other end of the channel performs a read, thus making metadata passing follow the program flow. Causeway invokes a meta-application through callbacks. The authors implemented a priority scheduler for a Web server application in only 150 lines of code, making a convincing argument for the feasibility of building meta-applications using OS support. A concern to be addressed in the future, however, is security and, in particular, the illegal modification of metadata by the running program.

Petros Maniatis suggested that Causeway could benefit from the use of both metadata passing and log-based analysis, since logs are streams and could be mined as they go by. Anupam didn't find the idea feasible, however. Doug Terry wondered what Causeway could do in the OS layer that it couldn't do in the middleware layer. Anupam clarified that by OS support he meant modifications to the OS as well as system libraries, but failed to answer the question exactly.

- ***Treating Bugs as Allergies: A Safe Method for Surviving Software Failures***

  Feng Qin, Joseph Tucek, and Yuanyuan Zhou, University of Illinois, Urbana-Champaign

  Bugs are inevitable, and they lead to system failures. Existing solutions such as rebooting, checkpoint-recovery, application-specific recovery, and failure-oblivious computing cannot recover from deterministic bugs. Yuanyuan Zhou offered a different approach to this problem: Since deterministic bugs are hard to cure, then "run away" from them, essentially treating them as allergies. This is achieved by changing the execution environment on demand upon soft failures. Essentially, the program is rolled back after each change is applied to the execution environment until the bug disappears.

  This method is developed by the Rx system in a comprehensive, safe, noninvasive, efficient, and informative manner. Sensors detect bugs before the program crashes, and changes include padding allocated memory to avoid overflows, allocating memory in an isolated location to protect against memory corruption, and, in the worst case, dropping user requests. However, all changes respect the application's API. While preliminary results on escaping deterministic bugs are more than encouraging, there are still several challenges for Rx, such as committing on the program's output (one reply to user) and the need for more powerful bug sensors. Yuanyuan emphasized that to be successful against deterministic bugs it is necessary to make the system nondeterministic.

  Aaron Brown asked how Rx handles a bug that is detected after output is sent to the user. Yuanyuan said that once the output reaches the user the problem is hard, but before the output reaches the user, techniques such as data mining could help prevent this. Aaron then asked how Rx handles concurrent requests. The reply was that requests are not serialized, but replays after checkpointing are. Brett Fleisch asked whether by "non-deterministic" she meant increasing the percentage of hidden bugs compared to deterministic bugs, and she agreed. Petros Maniatis (Intel Research) pointed out that changes by Rx might break down programmers' optimizations and asked how Rx would cope with that and ensure safety without programmer feedback. Yuanyuan's reply was that future work will include identifying common assumptions made by programmers and incorporating them into Rx. Armando Fox asked how Rx compared to failure-oblivious computing. The answer was that Rx is more general. Dug Terry asked how Rx prioritizes changes to the execution environment so that the effects are maximized. The answer was that, when possible, multiple changes are made simultaneously. Machine learning could be of further help there. Mary Baker asked how many rollbacks were necessary for avoiding bugs in general. The answer was not more than four.

SECURITY

Summarized by Nikolaos Michalakis

- ***When Virtual Is Harder than Real: Security Challenges in Virtual Machine-Based Computing Environments***

  Tal Garfinkel and Mendel Rosenblum, Stanford University

  Tal Garfinkel began by presenting functional differences between virtual and real (traditional) machines to support the hypothesis that such differences break existing security management approaches, so we have to rethink VM security. More specifically, traditional machines scale slowly and predictably while virtual machines do so rapidly, and traditional machines enforce homogeneity but virtual ones encourage diversity. In addition, traditional machines support stable populations, but virtual machines support highly transient ones, and the difference is more acute since virtual machines allow increased mobility, making it harder to link the VM to its owner.

  The solution proposed is to move security-related functionality out of the guest OS and into a ubiquitous virtualization layer. Such an approach will help decouple security and management from the structure of the guest OS.

  Margo Seltzer observed that this architecture resembles a microkernel, where the Trusted Computing Base is pulled out of the VM. Tal replied that microkernels were cool, and he didn't find anything wrong with that. Edward Wobber asked whether updating the VM state from outside the VM could be useful. The reply was that, depending on the OS, it could be. Jay Lepreau

followed up on Margo's remark, saying that there is no problem in separating the security from the VM. Tal added that such separation can give more control to administrators and more flexibility to users. Rik Farrow asked whether the security layer would look like an additional virtual layer. Tal mentioned that the platform needs to be beside the VM for security, but it is an interesting question what the actual architecture will look like.

- ***Make Least Privilege a Right (Not a Privilege)***

  Maxwell Krohn, Cliff Frey, Frans Kaashoek, and David Ziegler, MIT; Petros Efstathopoulos, David Mazières and Steve VanDeBogart, University of California, Los Angeles; Michelle Osborne, New York University

  Max Krohn said that a problem faced today by servers is that process boundaries do not always align with an application's security goals. Alice can steal Bob's data via buffer overruns, trojans, SQL injection, or even bad access control policies. Max presented a set of such scenarios based on Alice and Bob accessing the same Web site.

  To avoid these issues, Asbestos OS uses Mandatory Access Control (MAC). Asbestos uses compartments to track and control data flow. Unlike other systems, compartments are introduced not only by the kernel, but by applications as well. The data tagger, which is a small component that has no privileges, tags data based on users. When running an Asbestos Web server, data flow is tagged; the more the components of the application/system are touched by data without conflicting tags, the more a compartment grows, independently of the processes involved. Compartments are tagged upon reading data, and the more elements that are touched (e.g., processes, sockets, virtual memory pages), the more a compartment grows. If a compartment that is already tagged by user A is touched upon a read by data from a new user B, the compartment is tagged anew with a third tag, AB. This prevents data from being written out to compartments having tags A or B, thus protecting users' data from each other. When an operation is done, the tags on a compartment are removed and the components restored. The system finally uses trusted declassifiers that can act on behalf of multiple users and traverse subprocess boundaries.

  Philip Levis asked how Asbestos deals with database security. The answer was to have user data on different pages (serving as compartments) and a trusted index server. Jay Lepreau wanted to know how Asbestos differs from Flask in doing MAC in a distributed way. The answer was that applications, not only the kernel, can introduce compartments. However, as Jay noted, a trojan might contaminate a declassifier and get access to other users' data. Max agreed that they need to be careful with declassification.

  Pei Cao asked whether an attacker could trick the process into restoring a component. The answer was that only the kernel enforces the restore. John DeTreville asked whether this fine-grained control is better. Max said, "The more fine-grained, the better." Alex Snoeren asked whether a compartment tag could be illegally changed in the case of multi-threaded or event-driven applications. The answer was that once a compartment has been labeled it cannot be accessed by a flow of different tag. Margo Seltzer asked what happens with other data, such as registers. The answer was the registers are flushed similarly to a context switch. Peter Druschel asked what happens if user-specific data gets into a stack and another user finds it. One way to deal with this problem is to wipe the stack out when restore is issued.

- ***Access Control in a World of Software Diversity***

  Martin Abadi, University of California, Santa Cruz; Andrew Birrell and Ted Wobber, Microsoft Research

  Andrew Birrell described the first steps of a design for authentication and access control as part of Microsoft's Singularity operating system project. In actual operating systems the facts that principals are bound to either users or "logged-in" users and that ACLs are flat lists of principals are inadequate for making flexible access control decisions.

  The new design is based on three components: the naming tree, which records decisions the administrator or implementer has made (e.g., when installing a program), thus separating static policy decisions from online control ones; a compound principal mechanism; and a pattern recognizer for accessing control lists. In the Singularity design, the principal is just a string constructed by a path in the naming tree and logical operators. Since applications are part of principal names, they are described in the naming tree in the form of manifests. Andrew argued that enumerating principals in a control list will not give the desired flexibility. Instead, given a list and a principal, it must be determined whether the principal string is contained in the list string. The right approach, therefore, is to do pattern recognition, and for that reason regular expressions are used.

  Armando Fox asked whether time expiration is included in the design. The reply was that this might be useful but no compelling need was found for that yet. Margo Seltzer noted that regular expressions create a disconnect between flexibility of expression and usability, because they are not understood by mere mortals. Andrew replied that the ACLs will be created mostly by installation programs, not humans. Removing regular expres-

sions does not solve the problem. The goal is to be expressive. Michael Jones asked how reputation-based access control could be incorporated. Andrew replied that he would not like to add more complexity than that of the naming tree. Reputation makes him nervous. Michael Scott suggested that regular expressions could be used to find bad access control rules, something that Andrew agreed to look into. Alex Snoeren noted that since semantic value is put on the strings in the naming tree, there is a danger that if the tree is changed it will hurt the system. Andrew agreed that they better get these names right; relative paths would help there. Petros Maniatis asked whether they could do combinations of authentication methods in a scalable manner. Andrew replied that it was not possible in regular expressions; they would need to enhance their language.

SENSOR NETS

Summarized by Steve VanDeBogart

- ***PRESTO: A Predictive Storage Architecture for Sensor Networks***

  Peter Desnoyers, Deepak Ganesan, Huan Li, Ming Li, and Prashant Shenoy, University of Massachusetts

  Deepak Ganesan presented the ideas and motivation for PRESTO, a query architecture for sensor networks. Desirable features include low latency, low power utilization, the ability to query archival data, and the ability to formulate new queries after events have already occurred. PRESTO tries to provide all these features by taking advantage of the decreasing cost of storage as well as suppressing communications that report no new information.

  Many events that sensor networks are currently being used to monitor have domain-specific models. For instance, temperature variation is easily predicted from time of day and season. Based on previous data, the PRESTO proxy can send models to the sensor nodes. The sensor nodes can then only report events that violate the model. The proxy can then either send a new model or note the violation as an aberration. This technique is more energy efficient than a push model. It also allows the proxy to answer queries immediately, since the proxy is notified whenever an abnormal event occurs. If the query requires more accuracy than the model provides, the proxy will first examine its cache to see if it has already retrieved the needed information. If not, it will poll the relevant nodes. This information may be available at a lower tier in a multi-tiered network, possibly sparing the energy-scarce motes at the bottom from answering the query directly; if not, the data is cached at each tier on the way up, preventing further direct queries of the low-level node if the data is needed again. These techniques provide a middle ground between streaming out all the data, which is energy expensive, and querying nodes directly, which is slow.

- ***Towards a Sensor Network Architecture: Lowering the Waistline***

  David Culler, Prabal Dutta, Cheng Tien Ee, Rodrigo Fonseca, Jonathan Hui, Philip Levis, Joseph Polastre, Scott Shenker, Ion Stoica, Gilman Tolle, and Jerry Zhao, University of California, Berkeley

  Philip Levis began by saying that sensor network research today is a mess. There are a lot of different options for solving a given problem, but each solution is vertically integrated into a complete stack of solutions and each component in a stack is incompatible with any other stack. Therefore, if you want to use modules from more than one stack you have to build a totally new stack that integrates the components you need. This lack of capability is a limiting factor to the advancement of sensor networks.

  This wasn't a problem for the Internet because there was a well-defined protocol, IP, midway through the network stack that allowed work to proceed in parallel above and below that point. Can we just use IP in sensor networks? No, it isn't appropriate, but we should develop something for sensor networks that serves the same purpose that IP does for the Internet.

  Philip went on to argue that SP, the Sensor Protocol, should be a single-hop protocol that provides a richer interface than just send and receive. It should not specify the wire protocol, because the underlying link layer varies too much. It should work both for address-free protocols, such as flooding and tree collection, and for name-based protocols. There should be an interface for the layers above to specify a forwarding predicate. Furthermore, it is important that it provide interfaces for things that have to cross layers, such as power management, timing, and security.

  Petros Maniatis asked if SP will help in the wild or if it's an academic exercise. Philip responded that it will take place in both domains. In an academic sense SP will help us understand things at a deep level, but at the same time it will facilitate real code. Additionally, because sensor networks still exist in an isolated administrative domain, we may be able to iterate the design, unlike IP.

- ***Breakout Sessions***

  Summarized by Rik Farrow

  During the last portion of the HotOS workshop, the attendees split into groups that had been arranged by Margo Seltzer. The assignment for each group was to design and present a paper on a particular topic in one hour. As it turned out, all papers that included a PowerPoint presentation were accepted, and the one group that failed to reach that point had its topic rejected.

  Completed papers will become part of the HotOS 2005 proceedings.