conference reports

■ Our thanks to the summarizers:

Ashwin Bharambe

Christopher Clark

Priya Mahadevan

Tipp Moseley

Mohan Rajagopalan

Marianne Shaw

Alan Shieh

Craig Soules

Andrew Warfield

Charles Weddle

OSDI '04: 6th Symposium on Operating Systems Design and Implementation

San Francisco, California
December 6–8, 2004

DEPENDABILITY AND RELIABILITY

Summarized by Christopher Clark

■ ***Recovering Device Drivers***

Michael M. Swift, Muthukarup-pan Annamalai, Brian N. Ber-shad, and Henry M. Levy, University of Washington

Awarded Best Paper!

Michael Swift presented the first paper of the session on a technique for accommodating device driver failure.

Building on the previous Nooks work on protecting the operating system when driver failures occur, the next challenge addressed is how to keep applications running when devices fail. This is achieved by introducing shadow drivers tasked with masking failures by acting as a "spare tire" in the event of an emergency.

A single shadow driver is written for each device class, implementing the same interface to the operating system that a driver for a real device does. A shadow runs silently alongside each real driver, observing requests, until failure is detected by the OS. This triggers a restart of the real driver, while the shadow temporarily assumes responsibility for spoofing it and handling application requests until the restart completes. The shadow assists in reinitialization of the restarted driver by replaying previously observed configuration commands to the driver before handing back responsibility for requests.

This scheme was implemented in the Linux 2.4.18 kernel for sound cards, network cards, and an IDE disk driver. Evaluation used both artificial fault injection of common programmer errors and deliberate porting of real bugs into the test kernel. Results showed that 98% of errors examined were recoverable using shadow drivers.

George Dunlap (University of Michigan) inquired whether this work would reduce the inclination of companies producing drivers to bother removing bugs; Swift advised not to tell them we are doing this. Val Henson (IBM Research) offered praise and asked if there were plans to port the work into the mainstream Linux 2.6 kernel; Swift indicated not, given the current "grad student quality" of the code. A delegate from HP Labs likened the work to a dangerous condition for humans, where sufferers are unable to feel pain. Swift argued he would rather not experience the immediate consequences of driver faults, preferring instead to receive failure frequency statistics.

■ ***Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines***

Joshua LeVasseur, Volkmar Uhlig, Jan Stoess, and Stefan Götz, University of Karlsruhe, Germany

Motivated by the desire to reuse existing device drivers written for commodity operating systems with the L4 research operating system, Joshua LeVasseur presented a method of using virtual machines to achieve this. The benefits are clear: they comprise a very large body of code and have undergone testing that would have to be repeated were they to be rewritten, if it were even possible to do so.

The classic technique for driver reuse is to write "glue code," implementing the device driver API of the OS the driver was written for and performing translation to the primitives of the new OS. The difficulty is that the driver API is often loosely defined, very wide, and messy, entailing manipulation of arbitrary data structures within the original OS, which consequently must be simulated by the glue code. The alternative approach proposed is to use the original OS itself as the glue code by running it atop a virtual machine monitor and exporting access to the device to other hosted virtual machines. Client VMs run a stub driver that communicates with the driver VM to achieve the desired use of a device. Memory protection hardware ensures that DMA initiated by device drivers is intercepted by the VMM and translated to indicate the correct physical addresses as necessary; with a paravirtualized VM, this step may be elided. A single machine arbitrates access to the PCI bus, with other machines communicating with this to implement higher-layered drivers.

A performance evaluation of the reuse of paravirtualized Linux drivers demonstrates single-digit percentage decrease for network and disk throughput, with a cost of approximately double the CPU consumption of the native driver.

Bin Ren (University of Cambridge) asked how the CPU scheduler selects which VM to run. LeVasseur responded that scheduling needed tuning to match the drivers in question. Val Henson noted that running multiple OSes to support the drivers increases the amount of code that needs to be correct for the system to function. LeVasseur countered that reuse of existing drivers increases confidence that the drivers are correct.

- ***Microreboot—A Technique for Cheap Recovery***

George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, and Armando Fox, Stanford University

George Candea gave a talk on micro-rebooting, or restarting components within a system to purge any damaged state. The aim is to make reboot-based recovery fast in enterprise systems.

Candea's thesis is that improved availability can be achieved if it is possible to restart only the faulty parts of the system. This is an argument for making component restart possible and fast; to enable this, one must refactor code to move session state in a separate lump, distinct from the ephemeral state within the component. The session state alone may then persist between component restarts, and the micro-reboot can address the symptoms of software failure, such as transient exceptions, deadlocks, and memory leaks.

To evaluate the approach, a prototype auction Web service was constructed from J2EE components running on the JBoss platform. Since rebooting individual components is significantly faster than rebooting the entire JVM and has little discernible impact on availability, the fault detector need not be very accurate, because false positives are not expensive. This encourages a strategy of micro-rebooting aggressively, or periodic "micro-rejuvenation" to keep the system healthy.

A delegate from Georgia Tech noted that in real applications there may be significant dependencies between components, and so the restarting of a single component may require the restarting of almost the entire system. Candea replied that the common design patterns encouraged by the use of EJBs make such tight coupling infrequent. A delegate from Rice University asked about the complexity of the prototype and whether it was entirely in Java or included modifications to the runtime. Candea responded that the JVM was not modified, 200 lines of code were added to JBoss, and the remaining implementation was composed of managed EJBs.

AUTOMATED MANAGEMENT I

Summarized by Andrew Warfield

- ***Automated Worm Fingerprinting***

Sumeet Singh, Christian Estan, George Varghese, and Stephan Savage, University of California, San Diego

This work addresses the problem of identifying worm outbreaks as quickly as possible. Sumeet began by identifying the design space and clarifying the three main requirements of their approach: response time, granularity of containment, and deployment. Their system, dubbed "EarlyBird," aims for a response time on the order of seconds, contains worms at a precise content signature granularity, and is network (rather than end-host) based. The key challenges in developing such a system are processing and storage: at gigabit line rates, packets must be processed in 12 microseconds or less, and log data accumulates very quickly.

EarlyBird capitalizes on two properties of worms for fast identification: content prevalence, which involves the frequency of the substring, assuming that there is an invariant substring across all instances of a particular worm, and address dispersion, the property that such substrings will travel between a large set of hosts. Their approach uses fixed-length substring hashes to find common signatures. Hashing has the additional benefit of allowing value sampling, through only examining packets in a specified range of hashes. EarlyBird has been deployed for eight months in separate academic and ISP environments, and has found all known worms as well as several new ones.

Petros Maniatis from Intel Research asked about how EarlyBird would fare against polymorphic worms. Sumeet answered that most encrypted worms were easy to classify because embedded decryptor code is invariant. Concerning SSH tunneling, he proposed that encryption be made gateway-to-gateway rather than end-to-end. Regarding harder polymorphism, for instance NOOP insertion, he said that more investigation was required. Brad Karp (Intel Research) asked how many hosts would be exploited before prevalence thresholds were crossed, and mentioned the 30/30 rule (the address dispersion threshold of 30 sources and 30 destinations) described in the paper. Sumeet agreed that this issue needed more consideration but said that they had moved on to better methods than 30/30 since the paper was written. Someone from Microsoft Research asked whether worms could circumvent EarlyBird's detection mechanisms. Sumeet explained that value sampling was difficult to outguess. Someone from Rice University asked about worms over P2P. Sumeet answered that in general these were not a problem, but that in some specific cases, such as BitTorrent, there is a risk of generating false positives.

- ***Understanding and Dealing with Operator Mistakes in Internet Services***

Kiran Nagaraja, Fabio Oliveira, Ricardo Bianchini, Richard P. Martin, and Thu D. Nguen, Rutgers University

Operator-caused outages are a major problem in Internet services. The authors explored ways to provide "realistic virtual environment"-based support to help prevent such mistakes. The talk was divided into three parts: understanding operator mistakes, dealing with the problem, and validation of their results.

In order to better understand operator error, the authors conducted a study involving 43 experiments and 21 operators of varying levels of experience. Operators were asked to perform a variety of both proactive and reactive tasks on a model three-tier Web service environment, and the experiments resulted in a total of 42 operator mistakes. The highest categories of mistakes were those resulting in degraded throughput or inaccessible service, most frequently as a result of configuration problems. In order to prevent operator mistakes from happening, the authors developed a virtual environment in which system changes could be tested before they were applied. Their system could be run online, by "shunting" the request stream, or offline using traces. Finally, Kiran presented a validation of their prototype. In a set of live operator experiments using their environment, six of nine mistakes were caught before being applied to the production system. The authors also emulated the operators from the initial experiments and were able to catch 28 of the 42 mistakes that were observed there. The most frequently corrected mistakes were those of global configuration and starting the wrong version of a service.

Andrew Whitaker (University of Washington) asked if mistakes had been symptomatic of operator unfamiliarity with the model three-tiered service. Kiran acknowledged that this was worth considering in future work but that this was the best initial approach to the study. Jonathan Appavu (IBM Research) asked what other tools might be developed to help operators avoid mistakes. Kiran pointed out that the biggest problem is that many tools, such as configuration checkers, are very application-specific. While these tools are useful, he pointed out that the approach described in the paper worked with all applications and tested the actual results of operator actions.

- ***Configuration Debugging as Search: Finding the Needle in the Haystack***

Andrew Whitaker, Richard S. Cox, and Steven D. Gribble, University of Washington

Andrew began by describing his work as a different sort of debugger, one targeted at configuration errors. As an initial example he presented the issue of Mozilla crashing sometime after a set of extensions had been installed and observed that current approaches to the problem might involve googling for "mozilla crash" or reading help menus. Unfortunately, in many cases these will not provide a solution, and even reinstalling the broken application will not fix the problem. Their work aims to provide tool support to systematically identify the causes of this class of "worked yesterday, not today" (WYNOT) configuration errors.

Chronus is a tool that the authors have developed to isolate WYNOT errors. In Chronus, the operating system runs in a lightweight virtual machine above Denali, a virtual machine monitor that the group developed previously. Denali allows block devices used by the operating system to be made into "time travel" disks, logging a history of all past states of the disk. To diagnose a configuration error, the user provides a probe script that tests for the existence of the error. Chronus will then do a binary search across a specified region in the history of the disk, booting the OS and running the probe, finally returning a pair of disk images in which the probe results transition from success to failure. By examining the differences between these images, Chronus will provide the user with the specific block update that resulted in configuration error. Returning to his Mozilla example, Andrew observed that the search process finished while he got coffee, and it isolated the error to the specific extension that was crashing Mozilla.

Chi Zhang (Princeton University) asked whether Chronus would be able to identify problems that resulted from large logical disk transactions, for instance the installation of new software packages. Andrew answered that extra tools would be required to identify larger-granularity causes. Shinji

Suzuki (University of Tokyo) asked a follow-up question regarding the effects of buffer caches, which Andrew agreed were also a problem. A third question regarded issues such as spyware, in which failure might occur after a disk is changed. Andrew pointed out that this was discussed in more detail in the paper, but that spyware was a problem in general. Finally, David Oppenheimer (University of California, Berkeley) asked Andrew to comment on the difficulty of probe writing, especially for difficult-to-test issues—for instance, changing Mozilla's text from English to Japanese. Andrew pointed out that Chronus's contribution was to change a very time-consuming class of configuration debugging problems into software testing problems. He agreed that while probes could be difficult to write, he felt that they could be developed by experts and reused in many situations.

FILE AND STORAGE SYSTEMS I

Summarized by Craig Soules

■ *Chain Replication for Supporting High Throughput and Availability*

Robbert van Renesse and Fred B. Schneider, Cornell University

This talk, given by Robbert van Renesse, described chain replication, a system for high-throughput replication. Their system services two types of requests: updates and queries. The storage nodes are formed into a chain, and then updates are sent to the head of the chain and queries are sent to the tail. When a storage node sees an update it processes and then forwards it to the next server in the chain. Once the request has been processed by the tail, it is guaranteed to have executed on all of the storage nodes, and an acknowledgment is sent to the client. This same guarantee means that any queries processed at the tail will return data seen by all storage nodes.

In this system, a master maintains the chain membership. Node failure is handled by the master and the previous node in the chain coordinating to remove the failed node. Nodes may only be added to the tail of the chain, and their content must first must be synchronized with the existing tail. Once synchronized, the master and the tail coordinate to move the node into the tail. All new queries are then sent to the new node. Any lost requests must be resubmitted by clients (consistency is maintained via the requirement that all updates be idempotent).

The results of this work indicate that chain replication can provide higher throughput than most primary/backup systems. David Shultz (MIT) asked how the system handled network partitions to coordinate chain formation. He was told that the configuration of chains among masters is done using Paxos, which automatically handles such failures. Jay Lorch (Microsoft Research) asked if they had examined weak-consistency chain replication. He was told that weak-consistency chain replication performs identically to weak-consistency primary/backup.

■ *Boxwood: Abstractions as the Foundation for Storage Infrastructure*

John MacCormick, Nick Murphy, Marc Najork, Chandramohan A. Thekkath, and Lidong Zhou, Microsoft Research Silicon Valley

This talk, given by Lidong Zhou, described Boxwood, a toolkit for developing distributed storage abstractions. Today, distributed storage systems make use of reads and writes of blocks (or objects) and strict interface primitives that sometimes make it difficult to layer more complex abstractions (e.g., b-trees, hash tables) on top of them while still maintaining the same guarantees of consistency and scalability.

Boxwood provides several tools needed to create more complex data abstractions: a lock service, a logging service, a consensus service, and a replicated chunk store. The first three of these can be used by the application to provide consistency to the new algorithm. The chunk store provides a replicated virtualized address space to store data across a large set of machines, thus providing reliability and scalability of data storage.

Lidong then described an example distributed algorithm they built by taking an existing non-distributed b-link tree algorithm and hooking it into the Boxwood abstractions to provide a distributed version of the algorithm. With this new distributed b-link tree in place, they were then able to layer an entire distributed file system, labeled BoxFS, over the abstraction. Performance analysis of BoxFS indicates excellent scaling from two to eight servers, and better performance than a system running NFS over NTFS.

Erik Reidel (Seagate) asked about their performance graph that indicated they were getting 0.5 MB/s throughput using 5 to 40 disks. Lidong answered that the graph was supposed to show the effect of lock contention on concurrency rather than actual throughput. Erik then asked how many clients were used in these experiments. Lidong responded that they used two to eight mount points with one client per mount point.

■ *Secure Untrusted Data Repository (SUNDR)*

Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha, New York University

This talk, given by Jinyuan Li, described SUNDR, a system for ensuring tamper detection of files stored on untrusted data repositories. They began by describing a new kind of consistency: "fork consistency." Fork consistency guarantees that if a server provides two clients with different copies of the same file, that it can never again provide those clients with the same copies of that file. Also, the server is unable to tell the client that their change has not been applied once it has agreed to the change.

To achieve this, SUNDR stores a version vector with each file. This vector contains a version number for each client of the file. When a client makes a change to the file, it obtains the file and the file's version vector. The client can then compare the provided version vector with its current vector from its last access of the file. If its stored vector is not an exact subset of the provided vector (i.e., its version numbers are all less than or equal to the provided ones, and the version number for that client is identical), then it can detect that the server has broken fork consistency. Undetected modification of the version vector is prevented using signatures.

Emin Gun Sirer (Cornell University) asked about the possibility of a malicious client using replay attacks to make the server appear faulty. Jinyuan indicated that even a colluding server and client could not break the fork consistency. Algis Rudys (Rice University) asked at what data abstraction level this would be most useful. Should it be coupled with each NFS operation? Each block? Jinyuan responded that SUNDR currently works at the block level, but that the techniques could be applied to any of these levels.

DISTRIBUTED SYSTEMS

Summarized by Priya Mahadevan

- ***MapReduce: Simplified Data Processing on Large Clusters***

  Jeffrey Dean and Sanjay Ghemawat, Google, Inc.

MapReduce is a programming model with an associated implementation for processing extremely large input data sets. Along with the programming model, MapReduce also automatically handles fault tolerance, I/O scheduling, load balancing of input data set among various machines, and inter-machine communication.

The programming model is designed such that the input and output is a set of key/value pairs; the computation is expressed as two functions, map and reduce, both of which are user specified. On supplying the input key/value pair, the map function produces a set of intermediate key/value pairs. The intermediate key/value sets are then used by the reduce function to produce the output. An example of this type of programming model is counting the occurrences of a specific word in an input data file or files.

There are two kinds of programs, master and worker. The master program is special, in that it delegates tasks to the workers. Map Reduce also handles the following functionality:

- Parallel execution: Input data is split into tasks, and each task is executed on different sets of machines. Users can also specify a partitioning function for this purpose.
- Fault tolerance: Failures are detected using periodic heartbeats, and in-progress tasks are then executed on other machines.
- Dynamic load balancing: The master takes proximity of the workers into consideration (with respect to location of the input data) while assigning tasks to the workers.

In addition several refinements—skipping bad records, generating sorted output files, providing status pages that indicate tasks in progress, etc.—are provided by MapReduce. The performance was tested for two benchmarks, grep and sort, on a cluster comprising 1800 machines. In conclusion, MapReduce simplifies large-scale computations, and since it handles most of the parallelization and distributed systems internals, users without experience in parallel and distributed systems can use it effectively.

A member of the audience wanted to know of any task that could not be handled using MapReduce. The answer was join operations could not be performed with the current model. Someone else wondered how MapReduce differs from parallel databases. MapReduce data is stored across a large number of machines as compared to parallel databases, the abstractions are fairly simple to use in MapReduce, and MapReduce also benefits greatly from locality optimizations.

- ***FUSE: Lightweight Guaranteed Distributed Failure Notification***

  John Dunagan, Michael B. Jones, Marvin Theimer and Alec Wolman, Microsoft Research; Nicholas J. A. Harvey, Massachusetts Institute of Technology; Dejan Kostić, Duke University

Managing failures in a distributed application is a challenging task: one needs to maintain a lot of state, and handling cascading failures require handling many different cases. FUSE is a failure notification mechanism that addresses the above issues. FUSE is not a failure detection service; it requires the participation of applications to guarantee failure notification. Examples of applications that could benefit from FUSE include peer-to-peer storage, multicast trees, and content distribution networks. The advantages of using FUSE include guaranteed failure notification, convenient handling of all corner failure cases, and reduction in distributed application complexity.

Applications create a FUSE group by specifying the participating nodes, and FUSE guarantees that every member in this group will be notified whenever a failure condition affects this group. By creating a spanning tree among the group members, FUSE can guarantee failure notification; it does not need to monitor all the paths between all the nodes.

FUSE can tolerate arbitrary network failures and node crashes, but it cannot handle byzantine failures. Applications need to handle such failures explicitly. The FUSE API comprises three methods: CreateGroup, RegisterFailureHandler, and SignalFailure. The authors implemented FUSE over the SkipNet overlay, so that they could take advantage of the DHT's liveness checking properties. Using a DHT also assures low network costs even when there are many groups.

FUSE was evaluated on the ModelNet testbed. Evaluation metrics included group creation latency time, failure notification latency, performance under churn, and false positive rates. During Q&A, someone asked whether FUSE could pinpoint which node in the group failed rather than simply notifying group members about a failure. It turns out that FUSE cannot notify the exact node where the failure occurred. Someone was concerned about how FUSE could handle transient network failures such as a certain node failing and recovering before all the group members could be notified of the news of the failure. The speaker said there is no good way to handle such a situation.

- ***PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services***

Ming Zhang, Chi Zhang, Vivek Pai, Larry Peterson, and Randy Wang, Princeton University

Anomalies in Internet routing are common, and detecting them is a nontrivial task. The irregularities could reside in either the forward or reverse paths and are hard to isolate. The contributions of this paper include large-scale study and classification of routing anomalies and techniques for anomaly detection and isolation.

PlanetSeer combines passive monitoring with active probing. Probes are sent only during the period of anomaly, so there is low network overhead. Both modules use TTL change and *n* consecutive timeouts in TCP flows for the detection mechanism. The probing module is made up of baseline probes (when a new IP appears), forward probes (when a possible anomaly is detected), and reprobes (to find duration of an anomaly). By clustering nodes based on their geographic location and choosing a node in each group for probing purposes, the probing overhead is reduced. The authors also use traceroute from multiple vantage points to narrow down anomaly location.

Some of their results:

- The authors found approximately two anomalies per minute over a period of three months.
- Tier-1 autonomous systems (ASes) account for the least number of persistent and temporary loops, path changes, and outages, while tier-3 ASes account for the largest.
- Temporary loops have much longer hop lengths than persistent loops. Persistent loops either get resolved very quickly or stay for a very long period of time (> 7 hours).
- Outages occur closer to network edge, while path changes have a much wider impact.

One of the more interesting questions posed was whether any correlation was observed between the anomalies observed at the rate of two per minute. The speaker replied that while they did not explicitly look at anomalies correlation, his guess was that they were correlated.

NETWORK ARCHITECTURE

Summarized by Ashwin Bharambe

- ***Improving the Reliability of Internet Paths with One-Hop Source Routing***

Krishna P. Gummadi, Harsha V. Madhyastha, Steven D. Gribble, Henry M. Levy, and David Wetherall, University of Washington

Krishna Gummadi began by stating that recently proposed overlay designs (RON, Detour) for improving Internet path reliability were overly complex. This observation was supported by a detailed study using a Planetlab testbed to measure Internet path-failures. About 3000 different types of destinations (including commercial servers and broadband home users) were probed from Planetlab nodes. A failure was defined as three consecutive TCP RST losses in response to TCP ACKs combined with a traceroute failure to the destination. The observed path-failure rates were four per week for servers and seven per week for broadband hosts. Most paths witnessed at least one failure every week. Furthermore, last-hop failures for servers were infrequent, implying that unavailability of servers could very well be due to path failures in the network. The conclusion is that while failures definitely exist, they are uncommon and short, and mechanisms to overcome these should themselves be lightweight.

Gummadi went on to propose a new scheme called Simple One-hop Source Routing (SOSR), which achieves the above objectives. The idea is simple: Instead of using complex multiple-hop overlay routes, end nodes just utilize one intermediary node for "routing around" in case of a failure. Several measurements were performed to understand the utility of such an approach. It was found that in most cases, the failure could be avoided by using any one of a large set of intermediaries.

The authors showed that the random-4 strategy (picking four random intermediaries) provides most of the possible benefits. Notice that this scheme does not require any a priori probing either by end nodes or by the intermediaries, and hence is stateless. Gummadi concluded by stating that in spite of these positive results, it is unclear whether end users will be able to perceive performance improvements due to SOSR, since multiple orthogonal factors contribute to overall end-user perception.

- ***CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups***

  KyoungSoo Park, Vivek S. Pai, Larry Peterson, and Zhe Wang, Princeton University

Most of the previous studies of the DNS infrastructure have ignored the impact of local DNS name servers (LDNS) on performance. In this talk, KyoungSoo Park, using a comprehensive set of measurements, showed that client-side DNS (LDNS) failures are widespread and frequent and reduce overall performance and availability. LDNS servers belonging to several Planetlab sites (these servers are site-specific and not tied to Planetlab) were monitored for an extended period by issuing trivial local name lookups. While most of the lookups took minimal time to complete (as expected), a surprisingly heavy tail was observed for the lookup times. Furthermore, such delays were widespread, across several sites, and were frequent. The authors cited two principal causes for this effect: overloading of local name-servers due to heavy memory pressure, and lack of maintenance.

The authors propose an incrementally deployable cooperative DNS lookup scheme (CoDNS) with an aggressive adaptive timeout to overcome LDNS problems. The basic idea is to forward a name lookup to one or more DNS servers at other sites when LDNS is suspected of failing. The choice of which servers to contact is determined by their proximity to the querying server as well as availability. The authors showed that CoDNS is able to remove the heavy tail of lookup times and add an extra "9" to the availability of the local DNS infrastructure.

A few issues were raised during the Q&A session: Andrew Myers (Cornell University) worried that CoDNS reduced the security of an already insecure and critical infrastructure. David Oppenheimer stated that a simple tweak to existing DNS implementations (viz., adding *remote* secondary nam servers in configuration files, and reducing the default timeout values) would essentially provide the benefits of CoDNS.

- ***Middleboxes No Longer Considered Harmful***

  Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, and Robert Morris, MIT Computer Science and Artificial Intelligence Laboratory; Scott Shenker, University of California, Berkeley, and ICSI

Middleboxes are defined as entities interposed between end hosts (over the Internet) that perform more tasks than plain IP forwarding. Several such middleboxes (e.g., NATs, firewalls, caches) are in common use. However, because they violate the end-to-end principle, middleboxes are not in harmony with the existing Internet architecture, despite their clear practical benefit. In this talk, Michael Walfish presented an architectural extension to the Internet (Delegation-Oriented Architecture, or DOA) to accommodate such middleboxes.

DOA is composed of two fundamental primitives: First, each endpoint (middleboxes are also considered endpoints) has a globally unique topology-independent identifier called an EID; second, receivers and/or senders can *invoke one or more endpoints as delegates* for routing messages. By treating NATs and firewalls as such delegates, DOA can elegantly incorporate middleboxes into the overall design. Furthermore, the DOA permits the existence of new functionality, such as off-path firewalls where entities external to an organization can offer firewall services.

In order to implement these primitives, an infrastructure for resolving flat EIDs to physically routable identifiers (IP addresses) is essential. The authors present a global DHT as a promising candidate for such a resolution infrastructure. While security and performance of the resolution infrastructure have been addressed to some extent by the authors in the paper, Steve Gribble (University of Washington) argued in the Q&A session that the next critical challenge for DOA is investigating suitable mechanisms for maintenance and troubleshooting.

AUTOMATED MANAGEMENT II

Summarized by Marianne Shaw

- ***Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control***

  Ira Cohen, Moises Goldszmidt, Terence Kelly, and Julie Symons, Hewlett-Packard Laboratories; Jeffrey S. Chase, Duke University

Ira Cohen presented an approach for automatically inducing models of system performance; the technique requires little or no domain-specific knowledge, and therefore can be applied to a wide variety of systems.

The motivation behind this work is that we, as a community, have figured out how to build complex, large-scale network services; we've instrumented those services to capture a large number of diverse performance metrics. However, for any particular failure or event, how do we know which metric or set of metrics we should be looking at? Which metrics will not help in determining the root cause of the problem?

This work automates the analysis of this large collection of instrumentation data using Tree-Augmented Naïve Bayesian networks, or TANs. By capturing traces of both normal and anomalous events from an instrumented three-tier Web server, and combining that with instances of Service Level Objectives (SLO) failures, TANs are used to produce performance models. These models can be used to select the set of gathered metrics that correlates strongly with higher-level Web server performance.

In evaluating their approach, several key observations were made. Small sets of metrics are much better than a single metric at predicting system behavior; for the Web server workload, it was typically three to eight metrics. Because each metric in the set is associated with a particular system component, the set can provide assistance identifying the root cause of anomalous behavior.

Questions focused on how to use the approach. If you are interested in the dynamics of the system rather than a binary observation, could you still use this technique? Yes, if you could convert those dynamics into a binary classification. Is it possible to use this technique for prediction of input? The authors do not yet have sufficient experience to know what the generated models will look like.

- ***Automatic Misconfiguration Troubleshooting with PeerPressure***

Helen J. Wang, John C. Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang, Microsoft Research

Helen Wang presented PeerPressure, a mechanism for troubleshooting misconfigurations in modern, complex operating systems and applications. PeerPressure uses Bayesian statistics to compare the Windows registry of a misconfigured machine with a collection of Windows registries from other machines; the statistics can then be used to find the misconfiguration and fix it.

PeerPressure embraces the conformity of computer systems and their configurations, and the belief that most applications work correctly on most machines. When an application is deemed to be working incorrectly, its associated Windows registry entries ("suspects") are captured by the user and fed into PeerPressure. Suspects are canonicalized and statistically compared with the collection of sample Window registries to generate a ranking based on the probability that a suspect is misconfigured. PeerPressure uses this ranking to modify entries in the Windows registry one by one until the configuration problem is resolved.

Twenty real-world "troubleshooting" problems and a database of 87 machines' Windows registries were used to evaluate PeerPressure. The system was able to diagnose the misconfiguration problem in 12 of these 20 cases, and to significantly narrow down the set of possible misconfigured entries for the remaining eight.

To demonstrate the obscurity of various misconfigurations, Helen introduced and showed the consequences of a misconfiguration error in the Windows registry during her talk.

- ***Using Magpie for Request Extraction and Workload Modeling***

Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier, Microsoft Research, Cambridge, U.K.

Rebecca Isaacs presented the use of the Magpie toolchain for automatically generating models of a system's workload that can be used for performance debugging, anomaly detection, and capacity planning.

Magpie is designed as an online mechanism, so it must handle intermingled requests, unrelated operating system and application events, cross-machine interactions, and monitoring of resource consumption in a lightweight, non-obtrusive manner. Therefore, rather than tagging each request flowing through the system, Magpie uses an application-specific schema to correlate system events corresponding to the same request. A request parser uses this schema while processing an event log to correlate events. These events are then associated with a particular request using a technique called "temporal joins," which attributes events to the same request if they could have occurred during the same valid interval.

Magpie was validated against traces of synthetic workloads and shown to be feasible for a two-tier Web server and the TPC-C Benchmark Kit. Someone asked whether they had looked into its applicability to real-world systems yet. While they would like to look at large distributed systems, currently they have only looked at two- or three-machine systems; they need to scale out to larger systems. Magpie does require application instrumentation, so it will require effort to apply to existing systems; they have been evangelizing to try to get instrumentation added to products.

BUGS

Summarized by Mohan Rajagopalan

- ***Using Model Checking to Find Serious File System Errors***

Junfeng Yang, Paul Twohey, and Dawson Engler, Stanford University; Madanlal Musuvathi, Microsoft Research

Awarded Best Paper!

This paper, presented by Junfeng Yang, was about identifying file system bugs by using model checking techniques. These bugs are potentially destructive, but traditional testing techniques have been ineffective due to the exponential possibilities that one needs to consider. The talk described a file system model checker called FiSC, based on the CMC framework, which was effective in finding bugs that would otherwise have been very difficult to detect using static analysis techniques.

Of the several interesting components that make up the system, the talk dealt primarily with state reduction for the model checker. The checking process starts with some state and sees whether the state was encountered previously. Instead of a randomized approach for a state-space search, they advocate a guided search for their testing. Consistency checks are performed through an abstract file system that models the file system (e.g., for tracking topology), and this can be compared with the model to check for errors. To handle journaling file systems they resort to logging. This description concluded with the observation that checking could be made more thorough by downscaling and via canonicalization.

George Candea (Stanford University) asked about the modifications required to apply this system to identify bugs in databases. Junfeng noted that this may be easy to incorporate.

- ***CP-Miner: A Tool for Finding Copy-Paste and Related Bugs in Operating System Code***

  Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou, University of Illinois, Urbana-Champaign

Zhenmin Li described a technique to identify "copy-paste" bugs in operating systems by adopting a programmer's perspective rather than software analysis. Zhenmin noted that in principle this work was similar to, and was in fact motivated by, plagiarism detection tools such as MOSS and JPlag. While the software engineering community has taken a recent interest in identifying copy-pasted code, existing tools have several shortcomings, such as high cost, inaccuracies, etc.

The basic idea is to apply subsequence matching to identify code that has appeared at least twice, an idea frequently used in data mining. The algorithm is based on identifying frequent sequences, building a sequence database, and composing (joining) sequences within the database. This process is repeated several times. The talk also described an example where their technique was able to identify a "forget to change" bug—where the programmer forgets to replace variable names in a copy-pasted segment of code.

The first question was whether their tool was suitable for other large systems and if they had tried it out elsewhere. Zhenmin replied that while they had only tried it on small software benchmarks, it could be suitable. Another interesting remark was that comments can be very useful in identifying copy-pasted code. Finally, someone asked whether their system could mine CVS code repositories. Zhenmin replied that this was something they were currently looking at.

- ***Enhancing Server Availability and Security Through Failure-Oblivious Computing***

  Martin Rinard, Cristian Cadar, Daniel Dumitran, Daniel M. Roy, Tudor Leu, and William S. Beebee, Jr., Massachusetts Institute of Technology

Martin Rinard presented a very interesting and entertaining paper on a controversial new concept, "failure-oblivious computing," which differs from the traditional fail-stop philosophy used to build computer systems. The driving principle here is that programs are complex and should be able to tolerate localized memory errors. The talk began with a discussion of bounds violations in the standard C model. An empirical evaluation of five "failure-oblivious" programs was then presented by comparing these programs to their regular counterparts in the context of security, initialization, correct continuation, and ability to handle attack input. While failure-oblivious programs had some limitations, the results of this evaluation looked promising.

During Q&A, someone asked whether this meant that bugs should not be fixed and we should not bother about them. Martin's reply was that with failure-oblivious computing they are no longer bugs, so the program should do what it's doing. Zin Dong (Princeton University) pointed out that while this idea would be useful for some things, it may not be able to handle linked structures. Rob Pike (Google) mentioned that he did something similar with data mining, and it would be more comforting to know that all the failures were stored in a log. Margo Seltzer (Harvard University) asked Martin to compare this to sandboxing systems; Martin replied that this was much simpler. Dawson Engler (Stanford University) noted that it may not be possible to track race conditions this way. Another interesting question was what if an attacker knew that the application being targeted was failure oblivious. Someone pointed out that this approach could be very frustrating, especially in pinpointing bugs, since a program would continue even when you want it to fail. Jay Lepreau from Utah mentioned that this would be analogous to testing when optimizations are turned on.

WORK-IN-PROGRESS REPORTS

Summarized by Tipp Moseley

- ***pDNS: Parallelizing DNS Lookups to Improve Performance***

  Ben Leong and Barbara Liskov, MIT

Up to 10% of DNS queries exceed 2s of latency. To hide this latency, overlay networks of resolving nameservers are cached and queried in parallel. This results in a latency being the maximum latency of *N* queries instead of the sum of the latencies of *N* queries.

- ***Trickles: A Stateless Transport Protocol***

  Alan Shieh, Andrew Myers, Emin Gun Sirer, Cornell University

  Typical protocol stacks require resources, limit scalability, are vulnerable to DoS, and are barriers to migration. Trickles proposes to move all state to the client, via continuations, which are self-describing and encapsulate server state. This enables transparent failover, load balancing, and anycast services.

- ***Surviving Internet Catastrophes***

  Flavio Junqueira, Ranjita Bhagwan, Alejandro Hevia, Keith Marzullo, and Geoffery M. Voelker, University of California, San Diego

  In order to improve server uptime and protect important data from attacks from worms, data must be duplicated across different operating systems and configurations. This approach will differentiate exploitable flaws in software, and is successful in surviving past and even more aggressive worms at low cost.

- ***Honeycomb: Enabling Structured DHTs to Support High-Performance Applications***

  Venugopalan Ramasubramanian, Yee Jiun Song, and Emin Gun Sirer, Cornell University

  DHTs show great promise to run infrastructure services because they are self-organizing, failure resilient, and highly scalable. Honeycomb investigates the space-time tradeoff in caching data, and guarantees <1 hop average lookup performance while minimizing resource consumption.

- ***PRACTI Replication for Large-Scale Systems***

  Mike Dahlin, Lei Gao, Amol Nayate, Arun Venkataramani, Praveen Yalagandula, Jiandan Zheng, University of Texas at Austin

  - PRACTI focuses on several principles involving replication for large-scale systems: separate mechanism from policy, and separate data and control paths. This result is a universal replication toolkit with the following attributes:
  - Partial replication: an order-of-magnitude less bandwidth and storage space
  - Topology independence: reduced time taken to synchronize
  - Arbitrary consistency: improved availability in disconnected operation

- ***Shruti: Dynamic Adaptation of Aggregation Aggressiveness***

  Praveen Yalagandula, Mike Dahlin, University of Texas at Austin

  Shruti is a dynamically adapting, lease-based mechanism that adapts based on read/write history.

- ***MOAT: A Multi-Object Assignment Toolkit***

  Haifeng Yu, Phillip B. Gibbons, Intel Research Pittsburgh

  Heavy user accesses to shared files requires replication of data objects and files. The goal of such a system is high availability for multi-object accesses, and the key issue of the problem is replica assignment. MOAT is the first system to observe the importance of replica assignment, shows strong theoretical results regarding best/worst assignments, and implements a toolkit for replica assignments.

- ***Causeway: Operating Systems Support for Distributed Resource Management, Performance Analysis, and Security***

  Anupam Chanda, Khaled Elmeleegy, Nathan Froyd, Alan L. Cox, John Mellor-Crummey, Rice University; Willy Zwaenepoel, EPFL

  Causeway provides a general-purpose, distributed, multi-tier framework for scheduling, performance analysis, and security and access control. This project is motivated by solutions that exist for single-node systems, poor ad hoc solutions for multi-tier systems, and the lack of a general-purpose framework.

- ***PLuSH: A Tool for Remote Deployment, Management, and Debugging***

  Christopher Tuttle, Jeannie Albrecht, Alex C. Snoeren, Amin Bahdat, University of California, San Diego

  Fundamental abstractions of remote deployment include things such as abstract description language, resource discovery, resource allocation, host and environment monitoring, experiment deployment, and execution management. PLuSH is a framework of components that integrates these abstractions.

- ***Using Inferred Emergent Behavior to Automate Resource Management***

  Patrick Reynolds, Duke University; Janet Wiener, Jeff Mogul, and Marcos Aguilera, Hewlett-Packard Labs; Amin Vahdat, University of California, San Diego

  To automate resource management, we must find a system's emergent behavior from events and discover highly suspicious behavior that is different from a programmer's stated expectation, statistically anomalous, or a dominant source of delay. To find problem sources, applications are instrumented to infer a model of system behavior. Multi-resolution tracing starts with a black-box approach and then explores the benefits of additional information, resulting in more specific, more accurate information.

- ***Using Access Logs to Detect Application-Level Failures***

  Peter Bodik, University of California Berkeley; Greg Friedman, Lukas Biewald, and H.T. Levine, Ebates.com; George Candea, Stanford University

  Sometimes it takes months or years to detect a failure in Internet services. Based on the assumption that users change behavior in response to failures, a chi-square test of access history can detect anomalous activity.

- ***A Trust-Based Model for Collaborative Intrusion Response***

  Kapil Singh, Norman C. Hutchinson, University of British Columbia

  Most intrusion detection systems emphasize detection; response is limited to blocking part of the network. This approach temporarily stops the intrusion but does not cost anything for the attacker. If network components collaborate to identify the source of attack, they can defend against it by attacking the attacker. An attacker is identified by a proof of attack using router logs of activity.

- ***The Ghost of Intrusions Past***

  Ashlesha Joshi, Peter M. Chen, University of Michigan

  There is a window of vulnerability between the discovery of a bug and the application of its patch. An administrator may not know whether an intrusion occurred in this window. An approach to this problem is to use virtual machine replay and introspection to detect the triggering of the vulnerability.

- ***SoftwarePot: A Secure Software Circulation System***

  Yoshihiro Oyama, University of Tokyo; Kazuhiko Kato, University of Tsukuba

  SoftwarePot is a user-level middleware system that provides a virtual environment "pot." The system contains a private namespace of resources and a private file tree, and it can be mapped to a real external resource.

- ***Implementing an OS Scheduler for Multi-threaded Chip Multiprocessors***

  Alexandra Federova, Harvard University and Sun Microsystems; Margo Seltzer, Harvard University; Christopher Small, Daniel Nussbaum, Sun Microsystems

  Multi-threaded chip-multiprocessors lead to contention for L2 cache. Modifying the OS scheduler to co-schedule hand-picked processes can lead to increased throughput of 27–45% and a reduction in L2 miss rate by 19–37%. Processes are characterized and profiled by predicting miss ratios by randomly sampling how often certain memory locations are reused (30% overhead).

- ***Charon: A Framework for Automated Kernel Specialization***

  Mohan Rajagopalan, Saumya K. Debray, University of Arizona; Matti A. Hiltunen, Rick D. Schlichting, AT&T Labs Research

  Charon takes a holistic systems design to combine programming languages and OS design to improve both performance and security. Charon uses binary rewriting capabilities and static analysis to achieve a reduction in memory footprint while ensuring correctness. Potential applications include synthesizing kernels for specific targets (motes, routers, cell phones), QoS, adaptation reliability, configuration checking, and bug discovery.

- ***Java in the Small: Enabling Standard Java on Embedded Devices Through Customization***

  Alexandre Courbot, Gilles Grimaud, LIFL; Jean-Jacques Vandewalle, Gemplus Research Labs

  Many embedded devices would like to run Java, but often Java does not fit because the entire JRE is too large. JITS tailors a full-fledged JRE to a specific application based on runtime usage by removing unnecessary components and reducing space overhead.

- ***Singularity: Software Systems as Dependable, Self-Describing Artifacts***

  Galen Hunt et al., Microsoft

  Singularity is a new operating system developed by Microsoft to be used for dependable systems research. Dependability, defined as behaving as expected by creators, owners, and users, is the primary goal of this project. Singularity makes configuration a first-class concept with built-in abstractions. Online and offline inspection, verification using partial specifications, and IPC via bi-directional message channels are all supported.

KERNEL NETWORKING

Summarized by Alan Shieh

- ***Deploying Safe User-Level Network Services with icTCP***

  Haryadi S. Gunawi, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

  icTCP addresses the deployment of TCP/IP extensions. Many such extensions have been proposed in recent research. However, the transition from research to practice has been slow. Moreover, as new operating environments such as wireless networks emerge, new extensions may be needed. icTCP aims to reduce the kernel development costs of extensions by moving extensions from the kernel to user libraries, and adding a small, easy-to-implement set of kernel interfaces to enable multiple such user-level extensions. Thus, the kernel modifications are amortized over multiple extensions. Extensions written using icTCP require small amounts of kernel support, have low design and performance overhead, and are guaranteed to be TCP friendly.

  icTCP provides application read/write access to internal TCP variables (cwnd, ssthresh). By modifying these variables, applications can modulate the send rate. These variables are virtualized in that applications are not allowed to write arbitrary values, since this would enable TCP-unfriendly flows. Instead, only those transformations allowed by RFC 2581 are allowed, and so extensions are, by definition, TCP-friendly. The icTCP virtual variables should be applicable to most TCP implementations, since the TCP variables are found in most implementations. A recent packet history may also be provided; this extension is optional, since not all implementations keep such a history, and passing this history can be expensive.

icTCP requires 316 lines of code in Linux. The effectiveness and necessity of restricting the operations on virtual variables are confirmed. Multiple TCP extensions (TCP Vegas, TCP Nice, TCP-RR, TCP-EFR) were implemented as user extensions, at smaller line-number counts than the original kernel implementations. Extensions could be combined in a stack to leverage the benefits of multiple different extensions for the same connection. Interposing a user-level extension degrades performance slightly—bandwidth is not affected at small numbers of connections, but is slightly degraded at larger numbers of connections.

George Kola (University of Wisconsin, Madison) pointed out that TCP Reno has a coarse timer resolution, while TCP Vegas has a fine-grained timer resolution; he asked how icTCP supports TCP Vegas. The response was that icTCP has more fine-grained timeout. Andrew Whitaker asked how the icTCP technique applies to non-TCP protocols, for instance, congestion-controlled UDP. The response was that the authors have explored how UDP flows can use information from TCP flows and that this has not yet been implemented. Currently, the authors have only looked at algorithmic extensions, not new protocols.

- ***ksniffer: Determining the Remote Client Perceived Response Time from Live Packet Streams***

David P. Olshefski, Columbia University and IBM T.J. Watson Research Center; Jason Nieh, Columbia University; Erich Nahum, IBM T.J. Watson Research Center

Response time is critical, and poor response time can have economic consequences. Also, response time must be controlled to meet service level agreements (SLAs). Improving accuracy and minimizing latency to feedback (e.g., providing online rather than offline results) could improve the efficacy of automated management systems. However, existing methodologies have shortcomings. Probing at external points is either not scalable or does not have a high sampling rate. Application-level log analysis is typically offline and does not capture all the system latencies. Instrumenting Web pages requires overhead and changing the content, and does not work for all clients.

Ksniffer, a kernel-level latency analysis that captures the kernel and network latencies of a complete HTML page view, operates at gigabit rates on commodity hardware (e.g., relies on no driver modification and requires no special hardware) and works for all clients and all types of content, without instrumentation overhead. To minimize the persistent state of a packet, all packets are processed online, without intervening windowing or queuing. Using these techniques, ksniffer achieves low overhead while measuring response time accurately.

ksniffer returns page view information—the latency from the initial request of the root object, to the last object on that page. ksniffer does not parse HTML to identify the last object, since parsing is too slow and the HTML does not directly correlate with the actual fetch/processing order. Instead, ksniffer uses pattern learning to determine the embedded objects for a given page using referrer fields: The referral field for a request for an embedded object (e.g., a JPG or GIF) generally points to the container. These patterns are not used for situations where container information is directly available—e.g., requests from a single HTTP/1.1 connection, referrer field available. In the remaining cases, the referrer field is inferred by matching against the pattern cache. Where appropriate, low-level TCP latencies (e.g., propagation time for the last packet, connection setup time) are added to the page view time computed from this HTTP analysis.

The evaluation measured ksniffer under a range of experimental conditions. ksniffer results closely matched directly measured results from a modified client instrumented to directly report its perceived timeout. ksniffer correctly correlated the response time distribution within a subnet (similar distance from server), and differentiated the distribution between different subnets (different distance from server). ksniffer results also tracked the load surges in a highly variable stress test. Compared to Apache, ksniffer measured the correct response time, while Apache measured an order of magnitude lower (and incorrect) response time.

Ilya Usvyatsky (EMC Corporation) asked, “How do content distribution networks (CDNs) affect the correlation techniques?” The response was that the only way the CDN will affect the response time is if it generates the last completing download. However, since a CDN should be much faster than the server, and runs in parallel, the last download to complete is unlikely to come from the CDN. Stefan Savage (University of California, San Diego) commented that “often there are external objects, e.g., advertising banners, which could add significant overhead (especially when DNS is accounted for). Also layout and rendering time can dominate. These DNS/external fetches and layout issues are invisible to the server." The response was that the latency due to other Web sites can be significant, but if the critical path is not on your own server, optimizations on your server are not going to improve response time. Other tools are available for measuring end-to-end rendering. One can't measure this on the server.

- ***FFPF: Fairly Fast Packet Filters***

Herbert Bos and Willem de Bruijn, Vrije Universiteit Amsterdam, The Netherlands; Mihai Cristea, Trung Nguyen, and Georgios Portokalidis, Universiteit Leiden, The Netherlands

Code is available from http://ffpf.sourceforge.net.

FFPF reexamines packet filters, since the assumptions underlying their original design no longer hold. For instance, at the time, computational speed was close to network speed; this is no longer the case. While network monitoring is important, a large fraction of traffic is unclassifiable due to shortcomings in the expressiveness of traditional packet filters. Many monitoring solutions support only slow networks, or only sample portions of the input.

The goal of FFPF is to achieve high link rate without resorting to sampling. To allow more traffic to be classified, FFPF supports a more flexible notion of a flow as any packet stream that matches arbitrary criteria. FFPF is designed to support multiple simultaneous filters efficiently: common subexpressions of different filters are executed only once, and copying is avoided by allowing different filters to share buffers.

To minimize bus and memory bandwidth, operations are pushed as close to the data sources as possible (e.g., executing aggregation operators on a NIC or in the kernel, rather than on the CPU or user space, respectively). For instance, a FFPF pipeline to count the number of packets in a flow would both filter and perform the count. FFPF supports multiple languages, and compiles to user space, kernel space, and network processor code (IXP1200). FFPF is faster than existing libraries; packet loss is lower than pcap, and CPU utilization is slightly lower for a single filter and considerably lower for multiple filters with common subexpressions.

FILE AND STORAGE SYSTEMS II

Summarized by Charles Weddle

- ***Energy Efficiency and Storage Flexibility in the Blue File System***

Edmund B. Nightingale and Jason Flinn, University of Michigan

Edmund B. Nightingale's presentation began with a discussion of ubiquitous computing—specifically, network variability, power management, and stale data. This led to the introduction of the BlueFS and the "read from any, write to many" strategy. The BlueFS's flexible cache hierarchy extends battery lifetime through energy-efficient data access, supports portable storage, and improves performance by leveraging the unique characteristics of heterogeneous storage devices.

The presentation next discussed how BlueFS's implementation handles read from any/write to many, as well as power management, hiding device transitions, cache management, and cache consistency. The BlueFS implementation consists of a user-level daemon called Wolverine that handles reading and writing of data to multiple local, portable, and remote storage devices. It also contains a kernel module that intercepts VFS calls, interfaces with the Linux file cache, and redirects operations to Wolverine. In addition, there is a BlueFS server that stores replicated data. Most interesting is the ability of the BlueFS to hide device transitions to mask the performance impact of device power management. The BlueFS can also create device affinity so that the latest version of an object will always be cached on a particular device.

BlueFS compared favorably to NFS and Coda in a modified Andrew benchmark, being over ten times faster than NFS and 19% faster than Coda. The evaluation showed that, because of its ability to hide access delays caused by disk power management, BlueFS can read 4k-sized files up to 60 times faster then ext2 starting from a disk in standby mode, due to the ability of BlueFS to hide access delays caused by disk power management. Someone asked whether it was assumed that a connection to the network must be present. The presenter responded that if data consistency guarantees are wanted, then a network connection must be in place.

- ***Life or Death at Block-Level***

Muthian Sivathanu, Lakshmi N. Bairavasundaram, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

Muthian Sivathanu began with a discussion of how liveness information is not available in modern storage systems and how certain functionality can be enabled with this information: for example, eager writing, adaptive RAID, optimized block layout, intelligent prefetching, faster recovery, self-securing storage, and secure delete. This led into a discussion of how to make storage liveness-aware—specifically, through the two approaches taken by the authors, explicit notification and implicit detection.

Explicit notification adds new allocate and free commands to the existing storage interface. The file system is modified to use these commands to explicitly convey liveness information to the storage system. With implicit detection, the storage system monitors block-level reads and writes issued by the file system from underneath an unmodified interface and implicitly infers liveness information. The presentation points out that explicit notification is conceptually simple to implement but made difficult due to the asynchrony of file systems. Implicit notification can be implemented without an interface change but is fairly complex and ties the file system and storage system layers together.

The authors presented the secure delete case study they conducted to show the design, implementation, and evaluation of a secure deleting disk using both explicit notification and implicit detection. The authors chose the secure delete problem because it requires the tracking of generation liveness and provides a context in which liveness information is very important. For performance evaluation, a prototype-enhanced disk was implemented as a pseudo device driver in the Linux 2.4 kernel. Exploring the foreground performance of implicit and explicit secure delete, the authors found that the explicit implementation performs better. When asked whether the implementation duplicated file system functionality, Muthian stated that they duplicate a small amount but only on disk structures about the file system.

- ***Program-Counter-Based Pattern Classification in Buffer Caching***

  Chris Gniady, Ali R. Butt, and Y. Charlie Hu, Purdue University

  Chris Gniady began his presentation with a discussion of the buffer cache in file systems and how important the buffer cache is to performance. A key observation in process architecture is that program instructions, or the instruction's program counters, provide highly effective means of recording the context of program behavior. This led to the introduction of PC-based pattern classification (PCC). PCC identifies the access pattern among the blocks accessed by I/O operations triggered by a call instruction in the application. These pattern classifications are then used by a pattern-based buffer cache to predict the access patterns of blocks accessed in the future by the same call instruction. Chris noted that this is the first demonstration of program counter-based prediction used in operating system design.

  Chris went on to describe the PCC design and talked about the pattern classifications in PCC. There are three reference patterns that PCC uses to classify the instructions: sequential references, looping references, and other references. These classifications are used by PCC to manage future block accesses by a classified program counter. Chris then discussed the implementation of PCC, how PCC data structures capture the classifications of the program counters, and how this information is used.

  In evaluating PCC, the authors compared PCC, UBM, ARC, and LRU through trace-driven simulations. They found that PCC compares favorably to UBM, improving the hit ratio by as much as 29.3%, with an average improvement of 13.8%. PCC also outperforms ARC, with the hit ratio improving by as much as 63.4% and with an average improvement of 35.2%. Lastly, the authors found that compared to basic LRU, PCC results in an average of 41.5% reduction in the number of disk I/Os. With this disk I/O reduction, PCC reduces the average execution time of LRU by 20.5%.