;login:
OCTOBER 2013 VOL. 38, NO. 5
ELECTRONIC SUPPLEMENT

Conference Reports

USENIX ATC '13: 2013 USENIX Annual Technical Conference

San Jose, CA
June 26-28, 2013

Virtual Machine Implementation

Summarized by Kyungho Jeon (kyunghoj@buffalo.edu)

Optimizing VM Checkpointing for Restore Performance in VMware ESXi

Irene Zhang, University of Washington and VMware; Tyler Denniston, MIT CSAIL and VMware; Yury Baskakov, VMware; Alex Garthwaite, CloudPhysics and VMware

Irene Zhang started her presentation by explaining that checkpointing is similar to "suspend," but while taking a checkpoint, the virtual machine can continue its execution. Because checkpointing is used for fault tolerance, taking a checkpoint can be done quickly, in less than a few seconds; restoring from the checkpoint, although slow, hasn't been a problem; however, recent applications, such as dynamic VM allocation and desktop virtualization, require short restore latency because if restoring from a checkpoint takes longer than 10 seconds, starting from a new VM would be better. To support such new applications, a new VM checkpointing system for VMware ESXi, named Halite, was developed. Halite reduces time to restore a checkpointed state from 25 seconds to slightly more than a second.

First, Zhang explained the challenges. Restoring from a checkpoint state is hard because it requires bringing the entire state of a VM from disk to memory. Current VMware ESXi uses an asynchronous method that starts the VM and reads pages when faults occur. A drawback is that the VM will be unusable for a while. The performance degradation is due to the VM's memory being swapped inefficiently, so the restore process reads randomly from disks. Therefore, we should eliminate the faults to disks to avoid the performance degradation. Zhang's team worked on predicting which pages that VM might access when it restores and prefetches them, but they found it is difficult to anticipate a VM's behavior.

Rather than predicting the working set, the key idea of Halite is to predict access locality, which means predicting what memory pages the VM will access together. Once it can predict access locality, it can store the pages together on disk, by using "locality blocks" Halite introduced, so we can achieve spatial locality. This is more helpful than predicting working sets because access locality does not change much as VMs change.

To predict access locality of a VM during the restore process, Halite uses two techniques: accesses during the checkpoint process and guest address space. Halite predicts these during checkpointing, and the prediction is directly used to save memory pages into locality blocks. Halite also uses other techniques, such as compression, but this detail was not discussed in the talk.

As a result of new techniques, Halite could reduce time to restore from a checkpointed VM state with small overhead on the checkpointing process. Worldbench, which simulates Windows desktops, shows that Halite takes a little more than a second for restoring, but ESXi takes 24 seconds. Halite adds a few seconds delay for checkpointing, whereas ESXi spends more than 10 seconds. Zhang also demonstrated that the locality block alone reduces restoration time by half compared to ESXi, but to achieve a sub-second performance, compression is also required.

Mathieu Isabel (Global Excel) asked whether using SSDs was considered for storing and restoring the checkpointed states. Zhang answered SSD was tested and she thought it would provide some improvement, but wasn't sure how much. Albert Chen (Western Digital) asked whether using the benchmark is predictive rather than realistic and random. Zhang answered that the benchmark was used because of repeatability, and Worldbench simulates random behaviors of Windows systems. Abel Gordon (IBM Research) asked about using huge pages (2 MB) for backing VM memory pages. Zhang said using bigger pages is much more efficient and will improve performance. Mathias Payer (UC Berkeley) asked about how much active memory can be grouped in locality blocks. Zhang said that the paper uses working set estimates. Payer then asked about using hardware performance counters for more precise groupings. Zhang answered that it would be possible if a virtual hardware counter is implemented, but doing it just for checkpointing is not a good idea.

Hyper-Switch: A Scalable Software Virtual Switching Architecture

Kaushik Kumar Ram, Alan L. Cox, Mehul Chadha, and Scott Rixner, Rice University

Kaushik Kumar Ram presented Hyper-Switch, "a highly efficient and scalable software-based network switch for virtualization platforms." He claimed virtual switches are becoming performance-critical components in datacenter networks as the number of VMs a server hosts increases. In the future, Ram expects most network traffic will never leave a single physical server. Therefore, we need a high performance virtual switching component inside virtualized servers.

Current network architectures in virtualized environments are implemented inside the same domain, either the driver domain (Xen) or the hypervisor domain (KVM). Hyper-Switch changes the conventional design by moving the switch into the hypervisor, but retains device drivers running in the driver domain. The idea is to separate the control and data planes. The data plane is small

enough not to inflate the hypervisor, and thus it can keep the trusted computing base (TCB) small.

There are two benefits to this approach. First, there is no memory-sharing overhead because packet buffers are never directly shared between two VM domains. Second, it allows better I/O resource usage accounting.

In addition to the design choice, Ram described several optimization techniques used in Hyper-Switch. First, preemptive packet copying avoids a hypercall on the receiver side. Second, Hyper-Switch delays packet processing and batches multiple packets so that it amortizes packet-processing overhead across multiple batched packets. This technique is implemented on both the transmitter and receiver sides. Third, Hyper-Switch offloads packet processing to idle cores to increase concurrency. This scheme also considers CPU cache locality.

Hyper-Switch was implemented based on Xen and Open vSwitch. In the evaluation, Hyper-Switch performed 56% better than Xen and 61% better than KVM under TCP stream-based workloads. Hyper-Switch showed perfect scalability until the number of VMs reached four, and a slow but linear speedup until the number of VMs was eight. After that, the throughput was degraded because there were no more idle cores for offloading packet processing.

Emin Gün Sirer (Cornell University) asked whether Ram could describe how big the system's resulting TCB will be and how the TCB will look assuming all the VMs running on a hypervisor are using the network. Ram answered that the TCB size is determined by the size of the data plane and is not proportional to the number of VMs running in the system. He also said he thought the data plane is about 5% of hypervisor (Xen) code. Rik Farrow asked whether the performance gain is from avoiding memory copy; it was not, because Hyper-Switch does not avoid any memory copy compared to Xen. The gain is mainly because Hyper-Switch avoids context switches between the VM and VMM.

MiG: Efficient Migration of Desktop VMs Using Semantic Compression

Anshul Rai and Ramachandran Ramjee, Microsoft Research India; Ashok Anand, Bell Labs India; Venkata N. Padmanabhan, Microsoft Research India; George Varghese, Microsoft Research US

Ramachandran Ramjee began by explaining the benefits of Desktop VM migration: it offers an always local experience; when used in servers, allocating the VM near the location provides better response time; and "parking" a desktop VM on a server allows the user to turn the local desktop off when not in use, thereby saving energy.

But there are also drawbacks and challenges. Moving the VM is a problem because recent desktop VMs have a large amount of memory; for example, transferring 4 GB will take one hour and you will end up transferring 200 GB a month. Thus, both number of bytes transferred and time to migrate are important metrics.

Ramjee mentioned his assumption that disks can be synced independently, using an out-of-band synchronization mechanism, so the talk would cover migrating memory states. Input replay looked promising for desktop VM migration, but Ramjee found that the technique was not effective for modern operating systems due to the randomness generated by prefetching and address space layout randomization (ASLR).

Ramjee then presented evaluation results showing that zero pages were dramatically reduced in Windows 7, and the input replay technique resulted in many non-identical, non-zero pages. Other than replay, various techniques such as rsync and compression were tested by the authors, but they were not good enough.

Ramjee suggested a new technique, semantic compression. The idea—based on there being various types of pages, some more compressible than others—is that using different techniques for different pages will benefit memory state migration. There are various kinds of memory pages: heap, image, kernel, prefetched, and free pages. Each page has a different characteristic. Free pages are 80% compressible but actually don't need to be migrated.

First, MiG identifies pages. Free or zero pages are not migrated. The authors found 33% of pages are identical, which means MiG also does full-page matches. Next, image pages are actually from file systems, so MiG uses the context to do a "diff" because data coming from a file system might have changed. For example, library (DLL) or executable files are already in disks of both sides. MiG pre-loads them and builds a table of chunks.

When migration occurs, MiG does not send the page, but fingerprints the page and finds whether the other end has the page. If yes, then the receiver just needs to update a pointer. For heap and other pages, MiG has an intra-VM redundancy layer, which indexes entire VM state and finds a lot more redundancies than gzip compression can.

Finally, Ramjee briefly presented evaluations. There were two metrics: byte savings and migration time. The evaluation was conducted on Windows 7 VMs with 2–4 GB of RAM. MiG can reduce 40–60% of bytes compared to gzip. For migration time evaluations, MiG's compute time is almost the same to gzip, but transfer time is about half compared to gzipped pages.

Konrad Miller (KIT) asked why he can't drop all the pages originally loaded from disks, synchronize the disks, and load from the local disk. Ramjee answered that if some pages were loaded from disk, that information is hidden in the memory. But it will be problematic if that information is not transferred. For example, it is possible SuperFetch data structure points to unloaded, garbage pages after migration. Then Ramjee was asked again whether he can use ballooning. The answer was that there is a problem because the prefetched pages are not low priority pages.

Computing in the Cloud

Summarized by Yanfei Guo (yguo@uccs.edu)

Copysets: Reducing the Frequency of Data Loss in Cloud Storage

Asaf Cidon, Stephen Rumble, Ryan Stutsman, Sachin Katti, John Ousterhout, and Mendel Rosenblum, Stanford University

Asaf Cidon started by introducing the notion that data storage for cloud service is spread over thousands of servers. Due to independent server failures, one major goal of these systems is the tolerance to node failures. Asaf pointed out that the current randomization-based data spreading approach only tolerates independent node failures; it is not durable to correlated node failures such as rack power outages that cause multiple servers to be offline.

Asaf than introduced the term copyset, which is a node set that contains all replications of one data chunk. Losing access to one copyset means losing access to one data chunk. Asaf also reviewed some previous work in providing data availability when correlated node failures happen. These previous works limit the nodes that one data chunk can be replicated on and provide better durability to correlated node failures; however, it also increase the recovery time because it restricts the ability to recover data from multiple nodes.

Asaf presented their replication technique, Copyset Replication, which randomly generates multiple permutations of copysets of the same size. The data chunk is replicated to these randomly created copysets. By controlling the number of permutations, controlling the scatter width of a data chunk is easy. Thus trading off between the data availability and recovery speed is easy.

Following the talk, Asaf was asked about how Copyset Replication can be used with a system that can be dynamically scaled up. Copyset Replication also works in this scenario, but it will become less optimal as more and more nodes are added to the system. Asaf said they are studying this as future work.

TAO: Facebook's Distributed Data Store for the Social Graph

Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, and Venkat Venkataramani, Facebook, Inc.

Nathan Bronson started by introducing the social graph at Facebook, which is a data structure that saves user information and related behaviors. The social Web servers will query the graphs and render the page for users. Due to dynamic user page contents, the social graphs are queried frequently to serve the enormous request load of Facebook.

Facebook uses TAO to save the social graphs and support high volume dynamic requests. The backend of TAO is traditional databases. To speed up the read and write requests, a hierarchical memcached system is used. There are two levels of cache in TAO, the leader cache and the follower caches. The leader cache simplifies the synchronization of read and write operation within one site and across multiple replicated datacenters. This architecture allows easy scaling for social graph storage.

Someone asked Nathan about the load range on TAO. Nathan said the regular load is 1 MB, but there are also other loads with different sizes. Another question was about using TAO for workloads that have a lot of write operations. Nathan said that this system is more focused on workloads with more read operations; they have a different design to handle workloads with more write operations.

PIKACHU: How to Rebalance Load in Optimizing MapReduce on Heterogeneous Clusters

Rohan Gandhi, Di Xie, and Y. Charlie Hu, Purdue University

Because of datacenter heterogeneity and differences in nodes, different times are needed to process the same amount of data. Rohan Gandhi and his co-authors focused on MapReduce as a representative datacenter application. Previous work such as Tarazu showed that it is effective to mitigate performance heterogeneity by rebalancing the workload on different nodes. But Tarazu only brings limited improvement to MapReduce job performance.

Rohan then presented PIKACHU, a new workload rebalancing approach based on Tarazu. Their approach articulates the tradeoff between estimation accuracy and the wasted work from delayed load adjustment. The experimental results show that PIKACHU can significantly reduce the job completion time with well-balanced workloads.

Following the talk, Rohan was asked about the reshuffling in PIKACHU that is required by the workload rebalancing. Rohan briefly explained how data is collected by the reduce task and then transmitted during the rebalancing.

Flash-Based Storage

Summarized by Rohan Gandhi (gandhir@purdue.edu)

FlashFQ: A Fair Queueing I/O Scheduler for Flash-Based SSDs

Kai Shen and Stan Park, University of Rochester

Kai Shen explained that there is a problem with fair queueing in multi-tenant cloud systems. Existing schedulers use time slices to achieve fairness, but this approach fails on two counts: (1) different I/O operations require different amounts of resources and time, (2) time-slice-based schedulers show poor performance in terms of responsiveness when large number of tasks are contending for I/O resources. Furthermore, the existing I/O schedulers suffer from the list of problems resulting from underutilization due to idle applications and loss of spatial locality from switching.

To improve fairness and responsiveness in multi-tenant settings, the authors propose FlashFQ, a fair queueing I/O scheduler using throttled dispatch and anticipatory fair-queueing techniques. FlashFQ is based on the Start-time Fair Queueing algorithm,

which uses virtual time and expected resource usage. In FlashFQ, the authors use a throttled dispatch mechanism, where they monitor the progress of active tasks (applications that are performing I/O operations) and dispatch the I/O request only if the progress of the current task compared to the most lagging task does not exceed the threshold. Secondly, the authors use an anticipatory fair queueing technique to keep the task active even when there has been a period where no I/O request was issued by that task (deceptive idleness).

The authors have implemented FlashFQ in Linux and evaluated the performance against Linux CFQ, Quanta, FIOS, 4-Tag SFQ(D) on three flash-based storage devices (Intel 311, Intel X250M, and OCZ Verted 3). The authors used real (Apache server) and synthetic workloads with different I/O request sizes for comparison, and showed that FlashFQ achieves better fairness as it slows down read and write requests equally when under contention. They also showed similar results for I/O requests with different sizes. In terms of responsiveness, the authors demonstrated that FlashFQ has the lowest response time among other schedulers for synthetic as well as real workloads.

Someone asked about the size and pattern of the requests issued in parallel. Kai Shen answered that they have evaluated synthetic and real workloads that have such a pattern. A participant from UT Dallas asked about incorrect completion time predictions. Kai Shen said a bad prediction would produce an incorrect allocation of resources for requests and acknowledged that the design work required to address this question has been left for future work.

The Harey Tortoise: Managing Heterogeneous Write Performance in SSDs

Laura M. Grupp, University of California, San Diego; John D. Davis, Microsoft Research; Steven Swanson, University of California, San Diego

Laura Grupp claimed that packing for capacity in flash memory has a negative impact on performance and endurance. As a result, the worst-case performance (in terms of latency) of the flash writes is on a par with disks. In this paper, the authors make use of the observation that different pages in flash have different latencies by scheduling programs to pages that match according to their requirements. They call their system Harey Tortoise, as it matches high, hare-like performance while preserving high, tortoise-like endurance. Laura demonstrated how observed latency differs, depending on how many bits are already written: e.g., the latency is higher for the second bit compared to the first bit. As a result, there are fast and slow pages.

To exploit this variation, the authors added functionality to page-based Flash Translation Layer (FTL). In their design, the FTL holds three queues: emergency, external, background. Each queue is a set of operations yet to be performed. Each write in the three queues has a choice of slow or fast pages. To allow access to fast and slow pages, their design uses multiple write-points on each chip. Traditionally, there has been one write-point. This functionality is useful in handling the bursty workloads where fast pages are required.

Laura summarized by showing that the performance of their system is better than previous systems using five different traces. The traces differ in burst size and idle time. They compared the improvement in write throughput by using multiple write-points. On average the improvement is 34%. They also showed that their system can achieve better rate-matching.

Someone asked what fundamentally causes the slow and fast pages. Laura said that different speeds are caused by the way manufacturers store the bits on transistors. The first bit stored takes less time to write compared to the second bit, and first and second bits are stored on different pages, resulting in different pages with different write latency. Someone asked whether the traces were collected by the authors for this paper. Laura answered "yes!" and offered to share the traces, too. Someone else asked whether three bits per cell means three pages with different speeds. Laura said, yes—with more bits, there will be more pages with different speeds.

Janus: Optimal Flash Provisioning for Cloud Storage Workloads

Christoph Albrecht, Arif Merchant, Murray Stokely, Muhammad Waliji, François Labelle, Nate Coehlo, Xudong Shi, and C. Eric Schrock, Google, Inc.

Christoph Albrecht described Janus, developed at Google, which partitions the storage on flash and disk based on performance requirements. The problem of partitioning arises due to the performance, capacity, and cost tradeoffs between the disk and flash. The crux of their system is to determine cacheability, which determines how data storage is to be allocated between disk or flash. They named their system Janus (a Roman god of beginnings and transitions) as they use the sampled distribution trace from the past to determine whether to provide a flash-based storage and also the amount of flash-based storage for a given workload, so that the hits to the flash storage are maximized. Apart from describing the design of Janus, the authors also showed real trace characteristics from Google.

The design of Janus is motivated by the observation that the demand for a chunk of data varies based on its age. In Janus, the file or a chunk of data is added to flash when it is created (rather than when it is first accessed). Each workload is characterized based on the age and popularity, to calculate cacheability. This matrix and also the priority of the workload is solved as an optimization problem with the constraint of the flash capacity. When the data is present in the flash, it is flushed to disk based on the FIFO and LRU age of the data. The authors then show that this way of varying the amount of data partitioned on flash and disk can improve cost effectiveness.

Before concluding, the authors showed that Janus predictions on flash hits are achieved with real workloads, and hits are up to 76% higher than unpartitioned flash. Venkat Venkataramani from Facebook asked about how data is partitioned. Christoph answered that the greater the workload, the more opportunities to separate the data and increase optimization.

Miscellanea #1

Summarized by Ioannis Manousakis (jmanous@ics.forth.gr)

Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store

Christopher Mitchell, New York University; Yifeng Geng, Tsinghua University; Jinyang Li, New York University

Christopher Mitchell began by mentioning that InfiniBand is now mature and cost-efficient, and thus ready to take over from the Ethernet infrastructure in datacenters. He compared Ethernet and InfiniBand to show how the latter can increase the performance.

Christopher then introduced their own system, Pilaf, starting with some related work that appeared in NSDI '13. He described how Remote DMA can increase the throughput of GET/PUT operations and how this mechanism has been adapted in Pilaf. He mentioned potential problems resulting from their system and how to overcome them, and he covered races on concurrent PUT/GET operations. A performance evaluation demonstrated that Pilaf was able to achieve much higher throughput and lower latency than Redis and memcached while using fewer CPU resources. They concluded that combining RDMA (for read-only) with traditional message passing (for write) provides the best balance between simplicity and performance. An attendee wondered whether the authors had compared their InfiniBand implementation with an equivalent Ethernet version. Christopher answered no. Had they considered using transactional memory (TM)? They were not aware of any part where they could benefit from TM. Was the CRC that they used necessary? Yes, it was. Someone asked how the hashing mechanism worked. Christopher explained this to the questioner.

Lightweight Memory Tracing

Mathias Payer, Enrico Kravina, and Thomas R. Gross, ETH Zurich

Mathias Payer said that their primary goal was to make a faster memory tracing tool for binaries than the state-of-the-art, an implementation with a Valgrind binary translator, which was 50x slower than the original execution. Mathias discussed the foundations of memory tracing and the problems that arise. Mathias introduced the goals and the requirements of this work—flexibility, isolation, and performance—and described the implementation of the memory tracer and its architecture. The major benefit over the Valgrind implementation was the reduced memory allocation of his custom binary translator.

Mathias then discussed the evaluation process by giving the experimental setup and four configurations for the purpose of wider evaluation. Those configurations were incrementally more expensive in terms of resources. The quantitative evaluation in terms of performance essentially concerned the overhead of the memory tracer. First, Mathias mentioned the highlight: he achieved really low slowdown (3x) compared to Valgrind (50x); then he provided the memory overhead, which was less than 90% for unmodified x86 applications. He concluded by discussing related work, without citations, and mentioned some similar solutions, their performance, and architectural differences. Someone wondered whether the authors used the LLVM to do any source-to-source transformation in order for the memory tracer to work. The answer was no. The second question was how operating system pointers are handled. Mathias answered that the OS pointers are supported by converting the pointers from 64-bit to 32-bit. Someone asked whether the memory tracer supports the vsyscall (fast system call). Mathias answered yes.

Flash Caching on the Storage Client

David A. Holland, Elaine Angelino, Gideon Wald, and Margo I. Seltzer, Harvard University

David Holland started by mentioning that all the caches in storage services are resident on the server side. He said that he has investigated the use of client-side caches instead and discussed the architectural advantages he has found. Then he summarized the overall roadmap of their work, how they obtained their results, and described the general cache architectures, requirements, and parameters. David concluded from evaluation results that write-back policy does not matter when you have this type of client server cache and suggested that write-through cache has the same performance, is less complex, and thus is more appropriate for this role. He said that persistence is good to have but is not critical for performance. Someone asked what would happen if the application used synchronous writes; would the cache work? David answered that it would. What would happen if there were a workload skew? Dave replied that the fraction of I/O outside the workload was roughly 20%, so a workload skew should not happen.

Practical and Effective Sandboxing for Non-Root Users

Taesoo Kim and Nickolai Zeldovich, MIT CSAIL

Taesoo Kim gave two examples of sandboxing: OS jailroot and downloading an untrusted executable or file from the Internet. He explained in detail what would happen without sandboxing and what a sandboxing mechanism should do. Then he gave an overview of the design of his work (MBOX) and discussed its two basic components: the layered sandbox file system and the system call interposition mechanism.

Taesoo moved to yet another example of how to use MBOX to install a Linux package as a non-root user or to check the safety of this package if it comes from an untrusted source.

Taesoo mentioned the major performance advance of their implementation, which was the use of the SECCOMP/BPF system to track the system-call flow between the sandboxed application and the operating system. This particular tracing design allows the monitoring of only a subset of the available system-calls, in contrast to other solutions where they trace them all. Taesoo mentioned the slowdown of applications when running in the sandbox and highlighted the 0.1–20% overhead they observed. Someone wanted to know how to implement directory changes. Taesoo answered that he should check the paper for details. Another attendee wanted more information about related work. Taesoo said that he didn't have that information. The last question was whether privileged calls work with MBOX. Taesoo replied that, because the authentication is done with user-ID, the tool would work.

Data Storage Session

Summarized by Srinivasan Chandrasekharan (schandra@cs.arizona.edu)

TABLEFS: Enhancing Metadata Efficiency in the Local File System

Kai Ren and Garth Gibson, Carnegie Mellon University

Kai Ren first explained that small files are abundant and one of the challenges of the Linux file system is the management of metadata. He showed results of experiments with workloads dominated by metadata and small file accesses to show that even ext4, XFS, and Btrfs leave much performance improvement to be desired. One of the interesting benchmarks that Kai ran was creating 100-million zero-length files in one directory.

Kai proposed TABLEFS, which takes a flexible combination of two ideas from prior work. The first is to collate metadata and small files to reduce random reads to hard disk, and the second is to use a log-structured merged tree to reduce random disk seeks. In the evaluation of this approach, Kai showed a data-intensive benchmark with spinning disks and SSDs. He also compared TABLEFS with ext4, XFS, and Btrfs and in the results showed that TABLEFS could outperform the other file systems by 50% to as much as 1000% for metadata-intensive workloads.

In answer to a question about FS integrity, Kai explained that they were able to maintain data consistency. Kai didn't have an answer to a follow-up question about why SSD didn't fare well in a comparison to ext4.

Characterization of Incremental Data Changes for Efficient Data Protection

Hyong Shim, Philip Shilane, and Windsor Hsu, EMC Corporation

Philip Shilane presented this paper in place of Hyong Shim. He explained that protecting data on primary storage often requires creating secondary copies by periodically replicating the data to external target systems. The paper is based on analysis done with 100,000 traces from 125 customer block-based primary storage systems. The goal of the paper is to minimize overheads on primary systems and to improve data replication efficiency.

Philip explained that the traces were collected from enterprise customer sites. He also mentioned that the writes tend to be highly localized and showed the I/O properties of these traces. He said that there was a lot more information in the paper. He also compared the sequential vs random write I/Os and said that they selected the most sequential and most random I/O for analysis. The paper proposed the idea of a replication snapshot whereby the I/O overheads were lowered by about 20%.

On the Efficiency of Durable State Machine Replication

Alysson Bessani, Marcel Santos, João Felix, and Nuno Neves, FCUL/LaSIGE, University of Lisbon; Miguel Correia, INESC-ID, IST, University of Lisbon

Alysson explained how state machine replications work and why they're important. He touched on the topics of durability, maintenance, correlated failures, and how real systems implement durable SMR. He further showed that current techniques of implementing durable state machine replication must be supplemented by three key techniques—logging, checkpoints, and state transfers—but that these have a high impact on the performance of SMR even when SSDs are used.

Alysson proposed using parallel logging, sequential checkpointing, and collaborative state transfers to alleviate the issue mentioned previously. Alysson evaluated Dura-SMart against the BFT-Smart replication library and in summary mentioned that their contribution is the principled way to deal with durability overhead without breaking modularity.

During Q&A, someone mentioned that one would need some kind of state containment and would also need a perfect fail pass. Another mentioned that parallel logging would have issues. There was a discussion between the author and the commenters and the discussion was taken offline.

Estimating Duplication by Content-Based Sampling

Fei Xie, Michael Condict, and Sandip Shete, NetApp Inc.

Michael Condit mentioned that removing duplicates of the same block is important because it saves space, but predicting the cost/benefit ratio is hard. The authors tried to address the question whether one can estimate space savings before enabling deduplication.

Michael talked about the goals of the estimator: to have a small memory footprint and a minimal compute overhead, to maintain an up-to-date estimate, to provide accuracy guarantees, and to accomplish this by looking at a small subset of the data. He mentioned that the last goal had proven to be impossible, but they had achieved all the others.

Michael went on to show the algorithm they used and, in a flowchart, how the algorithm was implemented in the NetApp storage server. He further showed the accuracy of adaptive sampling and evaluated the effect on CPU usage and IOPS. He showed that there was no significant impact on latency at the client.

Someone asked whether content-based filtering caused one to lose blocks? Michael replied that that shouldn't be the case.

Virtual Machine Performance

Summarized by Cheol-Ho Hong (chhong@os.korea.ac.kr)

DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments

Dejan Novaković, Nedeljko Vasić, and Stanko Novaković, École Polytechnique Fédérale de Lausanne (EPFL); Dejan Kostić, Institute IMDEA Networks; Ricardo Bianchini, Rutgers University

Virtualization in cloud computing has several advantages including the decrease of operating costs, the isolation of misbehaving applications, and the migration of operating systems across physical machines (PMs). Today's virtualization technology tries to allocate CPU and memory resources fairly among accommodated virtual machines (VMs). In spite of this effort, performance isolation in term of the resources cannot be achieved sufficiently because different VMs located on the same PM may contend for the shared cache or generate inefficient I/O request patterns on the shared disk. Dejan Novaković called this situation performance interference and indicated that current solutions are not practical because they must trace the execution patterns of the co-running VMs for a long time prior to deployment. He emphasized the scalability issue that those solutions cannot deal with when lots of new VMs are deployed daily within the cloud.

To solve the problem within quick online activity, Novaković et al. used three approaches in DeepDive, a system for transparently identifying and managing performance interference in virtualized environments. First, the warning system performs early stage examinations that enable negligible overhead. The system gathers the required statistics of each VM from hardware performance counters and the hypervisor. The system then constructs a multi-dimensional space while using the statistics. Interference can be detected when current measurements are not located in the acceptable region in the space.

Second, the interference analyzer conducts thorough and exhaustive analysis when interference is detected by the warning system. The analyzer clones the suspected VM and runs it in a sandbox. The cloned VM should show a similar level of performance to the original VM in the absence of interference. Otherwise, the analyzer can pinpoint the governing sources of interference by utilizing the classic cycle per instruction (CPI) model. Finally, the VM-placement manager finds the best place to migrate the VM that caused interference by beforehand running a synthetic benchmark that resembles the behavior of the VM. Dejan showed that DeepDive infers interference between VMs with high accuracy, within 5% error on average, and makes an exact decision on VM placement within a minute.

An attendee asked about the complexity and overhead related to the sandboxing technique when the cloned VM interacts with its environments and other VMs. Dejan answered that a proxy plays a role in processing communications between the VM and environments by intercepting the traffic to/from the VM in the sandbox. Additionally, the proxy provides a cache for the results of requests in order to decrease overhead. Someone else asked about an undesirable situation in which a VM completely saturates some resource, and all of the trials of the VM-placement manager fail. Dejan said that the manager tries to find the best place for the VM by ranking the choices, but that case may happen. He said that the manager needs to be revised with a more sophisticated theory. Another attendee wondered whether the profiling technique could be applied to the heterogeneous sector of the machines. Dejan answered that the three-level approach of DeepDive can cope with the situation by considering the particular hardware architectures.

Efficient and Scalable Paravirtual I/O System

Nadav Har'El, Abel Gordon, and Alex Landau, IBM Research—Haifa; Muli Ben-Yehuda, Technion IIT and Hypervisor Consulting; Avishay Traeger and Razya Ladelsky, IBM Research—Haifa

Nadav Har'El started his talk by explaining why software-based I/O interposition in paravirtual environments is essential. When the host software can interpose on the guest's I/O, it can easily perform virtual machine (VM) live migration, security scanning, and virtual networking; however, I/O interposition suffers performance degradation because of the way a paravirtual I/O system works. In a paravirtual I/O system, the guest sends I/O requests by using shared memory in the hypervisor and notifies the hypervisor of the requests. The hypervisor then processes the requests and makes a reply notification to the guest. During this long activity, if the host machine has lots of I/O-intensive VMs, the competition between guests can occur in the hypervisor and cause a scalability issue. Additionally, the guest sending and receiving I/O events must perform several time-consuming exits that switch back and forth between the guest and the host.

To address the two problems, Har'El introduced ELVIS, an Efficient and scaLable para-Virtual I/o System. First, to improve scalability and efficiency, ELVIS utilizes dedicated cores for I/O processing. Each dedicated core then performs fine-grained I/O scheduling where one thread handles requests of many VMs. In this I/O scheduling model, I/O requests can be more fairly processed with a short delay by inspecting the request queues and the I/O activity. The inspection decides which queue should be processed and how long the execution should last. Second, to mitigate the overhead incurred by multiple exits, ELVIS removes the existing notification mechanism, developing two exitless I/O notification methods. For I/O requests from the guest, the dedicated I/O core polls the shared memory used for I/O requests. The guest then does not need to notify the hypervisor. For I/O responses from the hypervisor, polling by the guest can unfortunately cause unnecessary waste of a significant number of CPU cycles. Therefore, ELVIS adopts the Exit-Less Interrupts (ELI) technique by which the dedicated core directly injects a virtual interrupt into the guest. ELVIS can improve the performance by up to 3x

compared to the original paravirtual I/O policy when 14 I/O intensive guests are deployed.

Jean-Pascal from VMware wondered whether other solutions that remove the receive copy might work better than ELVIS. Har'El answered that in some particular cases ELVIS can show weak performance; that question, however, is not ELVIS' goal. Someone else from VMware asked whether the dedicated core is manually or automatically handled by the hypervisor. Har'El answered that for now the dedication is performed in a static way where ELVIS always dedicates one or two cores per I/O. He said that they are looking into how they can extend or allocate the dedicated cores based on the activity of the virtual machines.

vTurbo: Accelerating Virtual Machine I/O Processing Using Designated Turbo-Sliced Core

Cong Xu, Sahan Gamage, Hui Lu, Ramana Kompella, and Dongyan Xu, Purdue University

Cong Xu began his presentation by pointing out that server consolidation is the key factor that influences the cost of cloud platforms; however, Xu indicated that poor I/O performance in virtualization causes harm to the level of server consolidation. Due to the complexity of CPU sharing, I/O performance in virtualization is far from an acceptable level. He continued by explaining how I/O processing is performed in modern operating systems. When device interrupts arrive at an OS, the I/O data is synchronously copied to kernel buffers. User applications then copy the data in the kernel buffers to their spaces asynchronously. In a single OS, these procedures are executed promptly by giving preference to IRQ processing; however, in virtualization, the guest OS that has to receive interrupts may not be scheduled, and the delay of IRQ processing can affect I/O performance significantly.

Cong Xu et al. proposed vTurbo to decrease the IRQ processing latency in cases similar to that of a single OS. This approach is based on the traditional perception that smaller time slices utilized by the scheduler can significantly improve the I/O performance of VMs, whereas longer time slices satisfy CPU-bound workloads. To solve this dilemmatic problem, the approach designates a specialized core, named a turbo core, for IRQ processing of the guest OS. The core is a normal core in the system except that the hypervisor assigns short time slices (0.1 ms) to it. The hypervisor allocates longer time slices to other regular cores for CPU-bound workloads. In the turbo core, the synchronous part of IRQ processing of each VM is performed during 0.1 ms. Because the IRQ handlers of each VM are scheduled within a short time period, vTurbo can achieve an acceptable level of performance without hurting CPU-bound workloads that are executed on other regular cores. The asynchronous part of I/O processing is eventually performed when the VM running on the regular core is scheduled. vTurbo was implemented on the Xen hypervisor. Cong Xu showed that vTurbo improves the TCP throughput up to 3x, UDP throughput up to 4x, and disk write throughput up to 2x.

An attendee asked how vTurbo can improve the TCP performance when the application does not consume the data. Xu replied that vTurbo puts the received packets into the kernel buffer and generates ACKs for these packets. Therefore, the TCP flow between the client and server will continue even though the application is not running. The attendee commented that the kernel buffer might be filled to capacity. Xu said that the situation is not a scenario the authors fixed, but they have a plan to improve it. Someone from University of Washington asked about the context-switching overhead on the turbo core. Xu answered that the way of processing I/O in vTurbo is not different from that of a single OS on the physical machine. Although vTurbo has some overhead incurred by cache misses, he said that it is negligible. Should vTurbo have more dedicated cores when the number of VMs increases? Xu replied that it depends on the workloads, and even for five VMs a single dedicated core is still enough.

Packets

Summarized by Ayush Dubey (dubey@cs.cornell.edu)

Network Interface Design for Low Latency Request-Response Protocols

Mario Flajslik and Mendel Rosenblum, Stanford University

Mario Flajslik began by referring to recent efforts, such as memcached and RAMCloud, which have reduced software latencies to the 1 ms range. Such trends have shifted the bottleneck to the network interface, where latencies are in the order of tens of microseconds. The critical path in the network interface of today requires eight PCIe transitions for each successful transmit and receive, out of which six are synchronous. The proposed interface design for reducing latency in request-response protocols, called Network Interface Quibbles (NIQ), seeks to minimize the number of PCIe transitions, along with reducing memory allocations and deallocations.

NIQ provides a split interface for small and large packets. For small packets, which are defined as the minimum-sized Ethernet packets (60 bytes), both transmit and receive steps are folded into a single PCIe transition. In case of transmission, the entire packet is embedded into the packet descriptor; in case of reception, the packet is embedded in the completion entry. This mechanism also achieves the benefit of avoiding any host memory buffers, because the packet fits in a 64-byte wide cache line. For large packets, the traditional approach of using DMA to transfer packet data is employed. NIQ also makes use of a novel polling technique that avoids memory and instead polls directly over PCIe, with replies put in cache rather than memory in order to reduce overall notification latency. The authors found this design choice worked as a much faster option than traditional interrupt-driven network notifications, which have high latency overheads.

NIQ was implemented on a NetFPGA board and evaluated using an object store application that supports a simple GET-PUT interface. The latency evaluation measured a GET request

latency for varying object sizes (4–1452 bytes) against an Intel x520 network card. The same experiment was repeated for multiple NIQ configurations (interrupts vs polling, small packet optimization on/off, etc.). NIQ with interrupts gave the highest latency, on the order of 20 ms, and polling reduced these numbers by a factor of 2. The fully optimized NIQ configuration gave latencies as low as 4–5 ms, which represents and 1.5–2x improvement over the Intel x520. NIQ was also demonstrated to provide better throughput. Another set of experiments highlighted the importance of proper management of processor power states to achieve low latency. The authors describe a tradeoff between deep idle states (low power consumption, high response time) and active states (high power usage, low latency).

Jean-Pascal Billaud (VMware) asked which CPU mechanisms were being used to manipulate the cache lines. Mario replied that they were able to map memory in the NIC to CPU memory with the help of the CPU. Also, as the NIC kept processing more of the memory, it signaled the CPU to reuse some of the old memory. Someone asked whether the authors tried any simulations with frames larger than 1500 bytes. Mario replied that they did not, but their methods should probably scale up to handle jumbo packets. An attendee asked about how much NIC functionality is actually implemented on the NetFPGA board, and whether there was any reason that the evaluation was done only for a simple application that supports GET-PUT operations. Mario answered that the NIC is fully functional, and the only reason for using NetFPGA was because the authors already had a lot of Xilinx infrastructure. The reason for choosing a minimal GET-PUT application was twofold—they wanted to see latency numbers representative of their use case of memcached, and they also wanted a minimal application because any other application would likely introduce other sources of overheads unrelated to NIQ. Finally, Konstantinos Menychtas (University of Rochester) remarked that there was no mechanism to throttle a process, and asked whether there was any way to involve the operating system in the scheduling process. Mario replied that often the best way to achieve low latency was to avoid the traditional OS path, but another way might be to dedicate a few queues in the operating system for the network interface card.

DEFINED: Deterministic Execution for Interactive Control-Plane Debugging

Chia-Chi Lin, Virajith Jalaparti, and Matthew Caesar, University of Illinois at Urbana-Champaign; Jacobus Van der Merwe, University of Utah

Chia-Chi Lin presented his work on interactive network control plane via deterministic execution. He argued that, because the control plane is responsible for routing protocols, it is immensely complicated and a source of the majority of bugs in the networks of today. Moreover, current solutions involve either automatic debugging tools that only detect anomalies and require a lot of manual effort for pinpointing the source of the bug, or interactive debugging by deterministically replaying network logs, which does not scale or is inaccurate. Hence, a tool that can interactively and accurately debug production networks, as well as their debug logs, is the need of the hour.

Existing works categorize nondeterministic network entities as either external events, such as failure of routers or links, or internal events, such as exchange of messages between routers. This paper presents a library called DEFINED, which eliminates nondeterminism by logging all external events and manipulating internal messages, and provides a deterministic timer API. DEFINED includes two algorithms for network debugging, which are designed for production and debugging networks. In DEFINED-RB, each node speculatively executes events from the log, and independently determines the correct ordering using logical timestamps. If the execution is correct, the node delivers the packet, but otherwise it rolls back its state to a correct point and replays the events. This is made possible by forking a new control plane process before processing each network event, and sharing memory between parent and child process for signaling. The paper includes an optimized ordering function for eliminating cascading rollbacks. The other mode of execution, DEFINED-LS, facilitates interactive debugging by dividing the network execution into a series of logical steps. Each step involves an initial transmission phase in which the messages are delivered to the nodes, and a subsequent internal node processing phase. The evaluation of the system demonstrates that DEFINED-RB imposes minimal latency overheads, whereas DEFINED-LS was responsive enough that every single step command completed in less than one second.

Jon Howell (Microsoft) asked whether, in the common case, forking a new process for every correct delivery of a packet imposed a lot of overhead. Chia-Chi replied that the forked chains were not very long, and physical memory was shared between processes, which resulted in acceptable overheads.

Improving Server Application Performance via Pure TCP ACK Receive Optimization

Michael Chan and David R. Cheriton, Stanford University

Michael Chan presented his research on a simple kernel-level optimization in the TCP receive path for improving server application performance. The optimization was motivated by an example application of a video server transferring a large file via TCP to a client, which requires a series of small control packets (ACKs) from the client side. This was further illustrated by a similar experiment, in which the authors measured that about 99% of the packets were pure ACKs, which used up about 20% of the cycles. The reason for these overheads was the network driver was allocating socket and data buffers, traversing up the network stack, and then promptly deallocating the buffers upon realizing the packet was a pure ACK.

The authors presented TCP-PARO (TCP Pure ACK Receive Optimization), which provides a lightweight parallel TCP stack with a fast path for the common case of pure ACKs. The NIC is required to send the packet to the parser/demultiplexer module to parse the header. If the packet is a pure ACK, it is delivered immediately to the ACK processing module; if not, it is then sent along the traditional network stack. This method evades the memory allocation and deallocation overheads for the common case. The implementation required a change of only about 500 lines of code to the Linux kernel. The evaluation of TCP-PARO first showed that running the original experiment of large file transfer saved an average of ~15% cycles. A breakdown of these savings indicates that the majority were on the receive path of the network stack and memory allocation and deallocation. Similar trends were mirrored for the number of instructions per request. In the case where TCP traffic is not ACK dominated, TCP-PARO was still successful in saving a few cycles, and more importantly, did not cause any overheads. The authors also demonstrated that by enabling TCP-PARO, achieving near linear scaling for single-source multicast (TCP-SMO) is possible.

Christopher Stewart (Ohio State) asked whether any sensitivity analysis was conducted across TCP configurations by changing congestion windows, and would that affect the percentage of ACKs and, hence, the savings. Michael replied that no sensitivity analysis had been done, and congestion windows remained at the default TCP level. Moreover, changing congestion windows may have unwanted negative effects on the network throughput. Predicting whether performance would improve or worsen would be difficult, because this would change the number of ACKs kept in the queue. Jon Howell (Microsoft) commented that this was a good opportunity for optimization, and asked whether there were any other similar opportunities for building a fast path. Michael suggested syn floods. Another attendee remarked that TCP-PARO introduces a bump in the wire for all packets, and questioned what the overhead would be in case of UDP or any other mode of communication. Michael answered that even in the case of small packets, TCP-PARO imposes no overhead and in fact saves cycles. In the case of UDP packets, the small overhead of initial parser/demultiplexer processing is masked by larger overheads of cache miss. There was a final query as to when this optimization would be included in the official Linux release, to which Michael commented that he was unsure, but that the authors maintain patches.