Parallax: Managing Storage for a Million Machines

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

1 Introduction and Motivation

OS virtualization is drastically changing the face of system administration for large computer installations such as commercial datacenters and scientific clusters. A recent report by Gartner predicts that commercial use of virtualization will triple over the five year period beginning in 2004 [1]. While it is commonly held that OS virtualization improves the utility, manageability, and scalability of large-scale environments, we believe that it is not sufficient in itself. In this paper we argue that the next key challenge facing these environments lies in the dramatically evolving requirements for the management of persistent storage.

More hosts: Over the past few years, academic labs, server hosting centers, banks and other related organizations have moved firmly in the direction of centralizing compute resources into single facilities. Clusters especially have gained considerable momentum: academic installations of between 500 and 1,000 nodes are not uncommon and we are aware of several industrial installations of between 5,000 and 10,000 physical machines in operation today. In these environments, OS virtualization will result in a multiplication by between 10 and 100 of the number of active operating system instances; we have corresponded with several organizations who expect one million virtual node clusters within the next few years. Needless to say, each one of these hosts requires a system image to boot from.

More availability: Live OS migration [2] represents a qualitative shift in the management of these systems. Virtual hosts may be moved between physical systems while they run: this not only allows administrators increased freedom to service hardware but is also being explored as a mechanism for load-balancing in cluster environments. In order for a VM to migrate, its system image must remain available, mandating the location and access transparency of persistent storage.

More history: In addition to the benefits of physical separation provided by migration, several research projects have explored the benefits available through storing historical versions of VM state and allowing them to "time-travel". Revisiting these past states of a VM's execution has been used for intrusion detection [3], configuration debugging [4], and debugging for software development [5]. In extremis, it is foreseeable that enough historical state could be preserved to perform instruction-granularity replay through the entire life of a cluster. Such functionality would provide a complete set of forensic information and be of interest to highly-secure installations.

These three orthogonal issues each imply an increase in the scale of storage required for clusters of virtual machines. In this paper we propose *Parallax,* a distributed storage system which simultaneously provides different views on a single underlying block store. Parallax tackles the problems of management and scale for huge numbers of both active and historical system images in large cluster environments.

The nature of this new environment has led to two key design decisions that distinguish Parallax from previous systems. First, we observe that system image management is effectively free of write sharing, allowing us to easily exploit persistent caching for high performance, and to eschew the complexity of a distributed lock manager. Second, we capitalize on the nature of the virtualized environment to run an isolated Parallax server on each physical host, giving it control of local disk and allowing it to serve the set of local VMs directly. Parallax also uses block-level copy-on-write techniques to support both sharing and lightweight snapshots.

2 Design Space

An executing virtual machine requires a certain amount of persistent storage to hold a root file system, application data, swap files, and so on. Over time, VMs may wish to snapshot their persistent storage to allow backup, to deal with subsequent application or human errors, or even to allow "time-travel" as described in Section 1. Finally, there may be storage required for VMs not currently executing but which may be re-deployed in the future.

We unify all forms of persistent storage in a virtual server farm under the concept of a *virtual disk image* (VDI), the basic unit of management. A VDI represents the persistent state of a VM at a certain point in its execution, is accessible from any physical machine, and is stored in a redundant fashion to ensure high availability and durability. VDIs have human-readable site-unique names which facilitate the lifecycle management of virtual machines (e.g. deployment, snapshotting, suspension, time-travel).

It is quite reasonable to think of managing millions or even tens of millions of VDIs across a single cluster. In the following, we first discuss why existing techniques are inadequate, and then present our design for Parallax and how it addresses this challenge.

2.1 Yet another distributed storage system?

Storage systems have been one of the most exhaustively explored aspects of systems research over the past 30 years. Probably the most relevant state-of-the-art in cluster-wide image management is that of storage area networks (SANs). There are several current commercial offerings which tout "storage virtualization": systems that aggregate a set of storage servers into a single block-level substrate, and then allow this substrate to be divided up into individual volumes for export to network-attached hosts. Four important factors distinguish Parallax from these systems.

First, SANs are very expensive. Many, especially academic, environments will desire an alternative to expensive storage products. Furthermore, given that clusters are typically built from commodity systems, each housing a commodity disk, it is realistic to imagine a storage system that aggregates these disks. A virtualized environment makes this even more true given that the system-wide set of disks may be directly controlled using a set of per-host, isolated virtual machines. The challenge here is to provide the *manageability* afforded by SANs in this new environment.

Next, the scale that we are attempting far exceeds the capacity of any SAN that we are currently aware of. Fortunately there is an economy to this scale: we expect hosts to be based on a small set of original *template* disk images, and take advantage of the fact that common blocks may be shared across images. The underlying block store in our system will overlay common data where efficiency permits, allowing common blocks to be shared in many situations.

Third, the creation of new disk images is of critical importance to our scheme. Preserving historical images requires frequent run-time snapshotting of active OS images. A design goal that we are targeting is to be able to reasonably snapshot a running OS's disk and memory state every thirty seconds. Additionally, we anticipate that new virtual machine instances will generally be composed from existing templates, and so the duplication of VDIs is also important. A fundamental aspect of our design is in the management of per-VM block metadata, and providing fast primitives to fork and snapshot an active image.

Finally, we make the observation that write sharing is unnecessary in VDI management since at any given time, there is at most a single VM associated with a particular VDI. We take advantage of this fact to aggressively write-optimize our system, and achieve very high disk performance with considerably less complexity than is seen in systems using a distributed lock manager and lease-based persistent caching.

2.2 Parallax: Basic design

Our basic approach is to eliminate write-sharing, enable aggressive client-side persistent caching, seed the system with a small number of template images, use snapshot and copy-on-write to allow block-level sharing and use simple replication for high availability and durability.

The local storage on each physical machine is partitioned into a persistent cache for locally hosted VMs and a contribution to a pool of distributed storage shared by the cluster. The latter is managed by a service running in an isolated "Parallax VM" that presents a simple block device abstraction to each user VM and translates requests for the virtual blocks that are visible to the VMs into requests for physical blocks distributed throughout the cluster.

The service maintains a radix tree for each user VM to perform the translation. The cluster may initially be seeded with a set of well-known base images (Fe-

dora Core, FreeBSD, etc.) where each image has a complete radix tree mapping each virtual block to a 64-bit cluster-unique physical block identifier. Additional base images may be added at any time.

When a new virtual machine is created, the base image is forked by creating a copy-on-write clone of the base image's radix tree. The new root belongs exclusively to the new user VM, which can write to it without fear of conflicts. Owned blocks are modified in-place until the next snapshot, while blocks from the parent image, including nodes from the radix tree, are modified using a copy-on-write scheme. Each link in the radix tree contains a writable bit. At snapshot time, the root of the parent tree is copied and all links in it are marked read-only. This mechanism allows modified portions of the tree to be faulted in to the new snapshot tree as copy-on-write occurs.

Successive snapshots and forks can be performed with a single increment of the generation number, followed by copy-on-write for both block data and the metadata stored in the radix tree. Read-only sharing is achieved for all data derived from a common ancestor image, but coincidental redundancy, e.g., two user VMs install the same package on their respective virtual block devices and create duplicate blocks, is not exploited nor detected in this basic scheme.

Writes are committed first to the local disk in the persistent cache and then to the permanent replicas within the cluster. Both data blocks (parts of VDIs) and index blocks (parts of the radix tree) are persistently cached, with a subset of both also being cached in memory. The cache maintains both the virtual and physical block address for data blocks, hence avoiding the need to do the radix tree lookup for cache hits. Write-back is performed periodically, and is also explicitly triggered by the creation of a snapshot.

Physical blocks are stored across a replication group composed of storage volumes on other hosts. Each storage server explicitly manages block allocation for its volumes. A block write to a replication group receives the allocated block ids from each server in the replication group and combines these ids to build the global block id for the replicated block.

2.3 Parallax: Improved sharing

Block-level snapshots with copy-on-write semantics allow extensive sharing between virtual machines with a common ancestor, and between historical snapshots of a single VDI. Additional sharing of redundant content is possible within a single VDI and across VM images when blocks are indexed by content.

The basic design can be extended to collapse redundant blocks without changing the fundamental structure of the block store and without affecting read performance and semantics. As described, the basic system uses a radix tree to map the per-VDI block numbers to 64-bit universal block IDs. With the introduction of a distributed service mapping content hashes to universal block IDs, an extra step in the block write process can consolidate duplicate blocks.

Writes are made initially to the local persistent cache and a content hash is computed asynchronously. This keeps potentially slow operations like content hashing and collision detection out of the critical performance path. The hash is computed and the hash-to-block map is consulted to determine if the block is a duplicate. If it is then the existing block ID is stored in the radix tree; otherwise the block is written as in the basic design and the hash-to-block map is updated.

The level of indirection for combining duplicate content allows it to be a straightforward add-on to the base architecture with the same distributed block storage pool. The look-aside cache hides most of the performance impact for writes, and nothing changes for reads. Potential storage savings are obtained at the cost of the computational cost of computing content hashes and the storage and network overhead of maintaining the hash-to-block map.

2.4 Discussion

Parallax comprises a flexible and lightweight snapshot mechanism and a simple (and largely orthogonal) distributed block store for replication and enhanced availability. Provided that a sufficiently rich set of base images is provided, most of the sharing between different VMs and different generations of a single VM will be captured through common ancestry.

Duplicate content within a single image and duplicate blocks created independently in different images can be exploited by the use of content hashing. However this adds an additional mapping structure and associated computation and storage overhead: it remains to be seen whether the benefits outweigh the costs.

3 Prototype Implementation

To elucidate the design of our system, we have developed a prototype implementation over the past several months. This is not a complete implementation, but

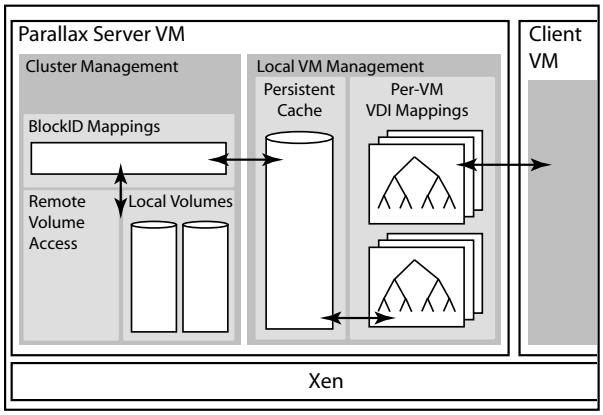

Figure 1: The Parallax server VM

serves as a proof of concept which uses the same data paths from VM to physical disk, and allows experimentation with the various design options and techniques that we have developed.

Our prototype extends the *block tap* [6], which is a block interpositioning mechanism for the Xen virtual machine monitor [7]. The block tap handles disk requests for a collection of virtual machines by forwarding them to a user-space library in an isolated VM. The tap maintains good performance while allowing us to easily modify the Parallax code.

The Parallax server is implemented as a user-space application in an isolated VM. In this configuration it is able to aggregate block requests from VMs on the local physical host and concurrently serve block requests from remote hosts. The Parallax VM receives direct physical access to local storage, and uses a GNBD[1] client library to access remote blocks.

The structure of our implementation is shown in Figure 1. The server currently implements a simple copy-on-write scheme, allowing remote GNBD images to be accessed by local VMs with writes stored on the local disk. While this implementation is considerably simpler than the full Parallax design, it serves to validate our approach and allow us to obtain baseline performance figures.

As shown in the figure, our prototype contains two points at which blockIDs are remapped. First, virtual IDs visible to VMs are mapped to a *logical ID* used by the cluster-wide block store. Second, these logical IDs are mapped to the physical hosts, disks, and blocks where the data is stored. In our prototype, this second mapping is one-to-one: VMs see the actual block addresses of a remote GNBD-mounted image. The first mapping, however, reflects the replacement of remote blocks in the VM's image with locally-stored copy-on-write blocks.

The intention of our prototype has been to guide design decisions and establish the feasibility of our approach for constructing a real system. To this end, we have measured the current performance, achieving remote read throughputs of 15MB/s to GNBD-connected images and 50MB/s to the local disk. Our implementation currently does not benefit from persistent caching, replication or parallel I/O, and uses a heavyweight mechanism to store the virtual to logical block mappings in lieu of radix trees. We are working on integrating these mechanisms into our prototype and anticipate dramatic performance improvements.

A further avenue of investigation involves the evaluation of the performance and functionality of our snapshotting and time-travel capabilities. As our design caters specifically to the frequent snapshotting of VDIs, we expect to achieve very good performance.

4 Related & Future Work

Distributed file systems have existed for over 30 years, and in common use since the late 80's. Most successful systems (e.g., AFS [8], NFS [9]) have in practice been 'networked file systems' in which one or a few servers export disjoint and non-replicated file systems to a number of clients. Many researchers have also proposed fully distributed file systems (e.g. Echo [10], xFS [11] and Farsite [12] to name but a few).

Our design is motivated by previous work on distributed block-level storage, most notably Petal [13] and the Federated Array of Bricks (FAB) [14]. FAB has recently also explored approaches to image snapshots [15]. Our assumption of single-writer access allows us to eschew much of the complexity present in these projects, we hope that this will allow us considerably more room to scale both in terms of number of images and frequency of snapshots.

Although we are not aware of any work directly addressing the same problem as Parallax, there are certainly similarities with other research. Frisbee [16] has explored the transport issues associated with efficiently deploying a template image onto the disks of a large number of clustered hosts. The notion of using an immutable store with copy-on-write stems back at least to Plan 9 [17], and similar techniques have been

[1] `http://sources.redhat.com/cluster/gnbd`

used by Elephant [18] and Venti [19]. Our current design is most similar to those from Bell Labs in that we do not consider deletes. However we hope to investigate ways in which deletion can safely be done, both to save space and to aid incremental addition and removal of storage devices.

In the future we hope to investigate how to most efficiently manage live migration [2] in the presence of aggressive persistent caching. A simple design would simply require write-back off all cached bocks for a particular VDI before a migrated VM can begin execution, but this could adversely impact VM downtime.

Instead we plan to keep LRU statistics for cached blocks on a per VM basis, allowing us to proactively transfer "hot" blocks to the destination node during live migration. Liaising with the guest operating system may also be of value, since certain blocks will already be contained within its private buffer cache. A further interesting question is whether we can choose the destination for migration based on the similarity of blocks cached at both locations; probabilistic similarity metrics such as bloom filters or sketches may make sense in this context.

Finally, we also intend to produce complete implementations of both the basic design of Parallax and the content-mapped variant, and perform extensive comparisons in terms of performance, availability guarantees, and sharing characteristics.

5 Conclusion

Virtual server farms and their variants are emerging as the architecture of choice for utility computing, and present a rather different set of distributed storage challenges. We believe Parallax represents a first step at addressing these requirements, and hope to see it evolve into the solution for these environments.

References

[1] T. Bittman. Predicts 2004: Server virtualization evolves rapidly. *Gartner*, November 2003.

[2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2005.

[3] S. T. King and P. M. Chen. Backtracking intrusions. In *Proc. 19th ACM Symposium on Operating Systems Principles*, pages 223–236, 2003.

[4] A. Whitaker, R. S. Cox, and S. D. Gribble. Configuration debugging as search: Finding the needle in the haystack. In *Proc. 6th Symposium on Operating System Design and Implementation*, pages 77–90, 2004.

[5] S. T. King, G. W. Dunlap, and P. M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proc. USENIX Annual Technical Conference*, pages 1–15, 2005.

[6] A. Warfield, K. Fraser, S. Hand, and T. Deegan. Facilitating the development of soft devices. In *Proc. USENIX Annual Technical Conference*, pages 379–382, 2005.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. 19th ACM symposium on Operating Systems Principles*, pages 164–177, 2003.

[8] J. H. Howard. An Overview of the Andrew File System. In *Proc. USENIX Winter Technical Conference*, pages 23–26, February 1988.

[9] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS version 3: Design and implementation. In *Proc. USENIX Summer Conference*, pages 137–152, 1994.

[10] A. D. Birrell, A. Hisgen, C. Jerian, T. Mann, and G. Swart. The Echo distributed file system. Technical Report 111, Digital Systems Research Center, October 1993.

[11] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In *Proc. 15th ACM Symposium on Operating System Principles*, pages 109–126, December 1995.

[12] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proc. 4th Symposium on Operating Systems Design and Implementation*, pages 1–14, December 2002.

[13] E. Lee and C. Thekkath. Petal: Distributed virtual disks. In *Proc. 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–92, 1996.

[14] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. Fab: building distributed enterprise disk arrays from commodity components. *SIGOPS Oper. Syst. Rev.*, 38(5):48–58, 2004.

[15] M. Ji. Instant snapshots in a federated array of bricks. Technical Report HPL-2005-15, HP Laboratories, 2005.

[16] M. Hibler, L. Stoller, J. Lepreau, R. Ricci, and C. Barb. Fast, scalable disk imaging with frisbee. In *Proc. USENIX Annual Technical Conference*, 2003.

[17] S. Quinlan. A Cached WORM File System. *Software Practice and Experience*, 21(12):1289–1299, 1991.

[18] D. Santry, M. Feely, N. Hutchinson, A. Veitch, R. Carton, and J. Ofir. Deciding when to forget in the Elephant file system. In *Proc. 17th ACM Symposium on Operating Systems Principles*, pages 110–123, 1999.

[19] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *Proc. 1st USENIX Conference on File and Storage Technologies*, pages 89–101, 2002.