ZoneFS: Stripe Remodeling in Cloud Data Centers

Lanyue Lu
University of Wisconsin-Madison
ll@cs.wisc.edu

Dean Hildebrand
IBM Almaden Research Lab
dhildeb@us.ibm.com

Renu Tewari
IBM Almaden Research Lab
tewarir@us.ibm.com

***Abstract*—Cloud data centers will contain tens of thousands of servers with massive aggregate bandwidth requirements for generating, accessing, and analyzing immense amounts of data. The I/O requirements of the myriad applications that these data centers must support run the gamut from extreme IOPS intensive to extreme bandwidth intensive. Delivering high performance with unreliable commodity hardware for this range of workloads is truly a grand challenge.**

ZoneFS is a parallel file system that targets cloud data center infrastructures built up of commodity network switches. ZoneFS employs a highly-available and flexible storage architecture that divides a cluster switch hierarchy into zones and stripes data across servers and disks to maximize aggregate I/O throughput and avoid storage server hotspots. In this paper, we present the overall design and implementation of ZoneFS and evaluate its key features with several cloud computing workloads. Our experimental results show that ZoneFS can improve application runtime performance by up to 76% over standard parallel file systems and by up to 85% over Internet-scale file systems.

I. Introduction

The emergence of cloud computing poses a grand challenge for modern data centers [1]. Cloud data centers will need to support thousands of simultaneous users, each running jobs that may generate and access petabytes of data with a variety of access patterns. Managing these massive data sets is increasingly becoming difficult, as no single file system can currently efficiently support every application. For example, file systems that support "Big Data" applications[2], [3] cannot support enterprise and web applications due to their lack of standardized interfaces, and enterprise file systems cannot scale to the degree required by "Big Data" applications.

As storage cloud architectures evolve to support more and more types of applications, file data must be globally accessible using standard interfaces. Carving up the data center into several specialized clusters, each with their own storage solution and access mechanisms, limits flexibility and will become unmanageable as no single client is capable of managing all the data. A single storage solution that can deliver good performance for analytic, web server, database, and virtual machines will be required. Moreover, cloud data centers use commodity hardware, and so any storage solution must be robust under failures. Consider a typical compute cloud architecture that presents a virtualized environment. Applications run inside a virtual machine (VM) and access data from a virtual LUN, which is typically stored as a file, e.g., VMware .vmdk file, in the storage system. If the virtual LUN is stored on a single server, the VM must run on that same machine, creating possible hotspots. Alternatively, if the virtual LUN is stored on a separate NAS server, then the VM can run on any server, but must transfer all of its data (greater than 80 GB) over the oversubscribed data center network. A better solution would stripe all file data across the servers connected to a single switch, allowing VMs to run on any server without having to transfer data across higher-level switches. This enormously simplifies both the time and complexity of configuring new VMs and dynamically migrating them from one server to another.

978-1-4577-0428-4/11/$26.00

Parallel file systems that perform wide striping have enabled supercomputers to push the limits of modern compute and data processing capabilities. Unfortunately, this approach is expensive and relies on a massive network switch. By forcing all I/O through a single switch, parallel file systems and other network attached storage solutions are severely bandwidth limited in cloud data centers that use cheap commodity-based compute clusters and networking hardware [4]. With each successive level in the data center switch hierarchy, more and more compute nodes must share a decreasing amount of available bandwidth to the parallel file system.

To take better advantage of commodity hardware and work around switch limitations, Internet scale file systems such as HDFS [5] and GFS [6] have applications use customized interfaces to access data locally by randomly assigning large data chunks directly on compute nodes. These file systems centralize large chunks (128 MBs) of contiguous data on a single node. This can create data hotspots as compared with parallel file systems that tend to use a 64K-1MB stripe size. While triplication helps alleviates these hotspots somewhat, it provides little benefit if an entire rack of 20 to 40 servers wants to access the same piece of data. In addition, re-replicating the terabytes of data on a failed node could take hours, if not days, increasing the probability of data loss. Another issue is that general applications have difficulty using these file systems since they do not support POSIX and remote data access via NFS or CIFS is slow. In addition, the tight coupling of data and compute ignores the required ratio of storage space to compute power in a data center. Administrators cannot just add storage or computing resources, but must scale both in tandem, which is particularly problematic for large data sets that need to be accessible but are infrequently accessed.

A. Contributions

In this paper, we propose ZoneFS, which explores empowering the file system with an understanding of the data center's

network infrastructure. ZoneFS views the data center as a collection of mini-clusters, or *zones*, each with their own high-bandwidth and fully connected switch. In this manner, ZoneFS uses parallel I/O to effectively load balance I/O requests across each zone, improving I/O performance over the entire range of I/O workloads.

ZoneFS is a novel POSIX-compliant parallel file system that retains the reliability, scalability, and performance benefits of standard parallel file systems while leveraging the commodity switch architecture of Internet scale file systems. ZoneFS distributes data across any number of nodes, eliminating data "hotspots". As the number of zones and data sets increases, so does the aggregate I/O bandwidth. Numerous applications can be launched on any number of nodes in a zone and all realize similar performance–even if they all access the same data sets. This paper uses benchmarks and applications to demonstrate that ZoneFS can exceed the performance of both parallel and Internet-scale file systems.

ZoneFS has a flexible data architecture that can support directly-attached disks, storage area networks, or even SAS disk arrays, with each offering different levels of cost, availability, and performance. For example, ZoneFS can use client-driven RAID across multiple storage nodes connected to SAS disk arrays. This allows clients to fail without affecting data availability and avoids the need to replicate data within a zone. In addition, by allowing a separation between compute and storage nodes, administrators can turn off under-utilized compute nodes for power conservation. In this paper, we analyze and compare directly-attached disk and Fibre Channel disk array architectures.

The remainder of this paper is organized as follows. Section 2 reviews the data center networking architecture and discusses the limitations of using parallel and Internet scale file systems. Sections 3 and 4 describe the ZoneFS architecture and our Linux prototype. Section 5 reports the results of experiments with micro-benchmarks as well as analytic and general applications. Sections 6 and 7 discuss related and future work. We summarize and conclude in Section 8.

II. Background

To better understand the motivation for ZoneFS, let's review current data center network topologies and how they impact modern file system architectures.

A. Data Center Network Topologies

Many data centers use multi-tier trees of network switches or routers [7]. As shown in Figure 1, servers connect directly into the leaves at the bottom tier, which consist of smaller GigE switches, e.g., 48 ports. The upper tiers aggregate the leaves to create a fully connected system. While the upper tiers may use 10 GigE switches, the network infrastructure is oversubscribed. Typical data centers are oversubscribed by a factor of 8:1 or even larger [7]. While it is possible to eliminate oversubscription and use a ratio of 1:1, it is cost prohibitive for most data centers. This oversubscription creates the inter-switch bottleneck that constrains data access in data centers.

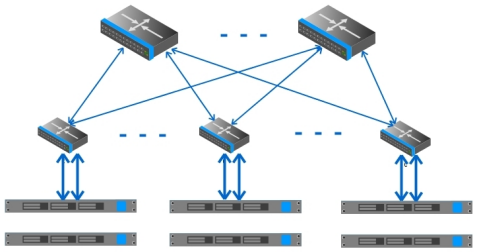

Fig. 1. **Data Center Network Architecture.** *Example 2-tier network architecture. Servers connect to leaf switches while upper tier switches create a fully connected network.*

B. Parallel File Systems

Over the last decade, parallel file systems such as GPFS [8], Panasas [9], and Lustre [10], have achieved unprecedented scalability in numbers of clients, servers, and disks. These achievements require that a system scale every aspect of its design–especially the network switching fabric. Parallel file systems increase aggregate I/O throughput by striping data across possibly hundreds of storage nodes. This technique can reduce the likelihood of any one storage node becoming a bottleneck and offers scalable access to a single file.

One example is the BlueGene/P Intrepid supercomputer shown in Figure 2, which connects 40 racks of compute servers to 640 I/O servers, which in turn are connected over a Myri-10 GigE switch complex to 128 storage servers. While cloud data centers can require such bandwidth, they lack the budget for such switches, and so it is doubtful that parallel file systems will emerge as a popular solution.

C. Scale-Out Network Attached Storage

Scalable NAS systems [11], [12], [13] use standard filing protocols such as NFS and CIFS to scale I/O throughput. Currently, many users of scalable NAS systems focus on IOPS, e.g., creates per second, rather than sustained I/O throughput [14]. Consequently, administrators can oversubscribe their data center network infrastructure with little consequence, since file create and stat operations tend to require little raw bandwidth. If these users start executing "Big Data" applications, they would encounter the same network oversubscription problems as with using parallel file systems.

D. Internet Scale Distributed File Systems

Users of "Big Data" applications have realized that POSIX-compliant file systems and their associated semantics can be overkill for their specific requirements. This has driven the creation of file systems that co-locate compute and data on a single node with directly-attached disk [5], [6]. While local disk access can increase I/O throughput beyond the bandwidth of the network link, it limits the number of applications that can access a single piece of data. In addition, co-locating compute and data is not always possible in a loaded system, and so inter-node data transfers continue to occur.

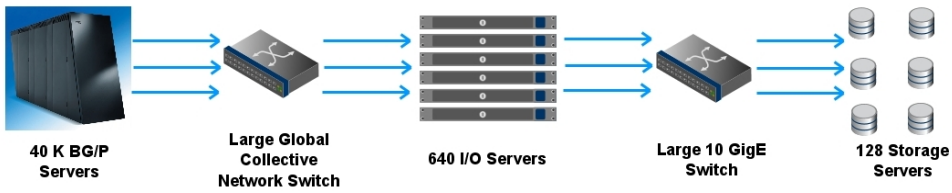

Fig. 2. **Intrepid Supercomputer and GPFS.** *40 BG/P racks redirect I/O requests over a specialized network to 640 I/O servers, which in turn use a large 10 GigE switch to access 128 storage servers.*

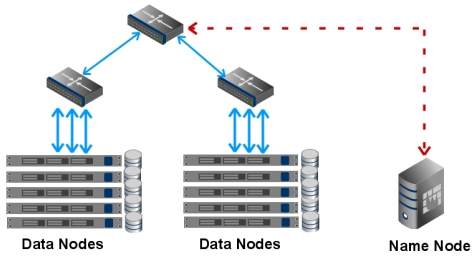

Fig. 3. **HDFS Architecture.** *Data is striped in large chunks (64-128 MB) across the disks directly attached to compute servers. A single NameNode is used to manage file system metadata.*

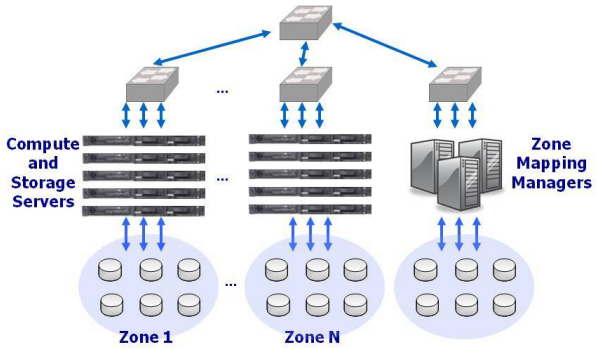

Fig. 4. **ZoneFS Architecture**. *The file system is made up of a set of zones, each consisting of compute and storage nodes connected to a switch. All data created within a zone is striped across the storage nodes in the same zone. A data-to-switch mapping manager tracks the allocation of data to each individual zone.*

III. ZoneFS

In this section, we introduce and describe ZoneFS, a novel parallel file system that uses parallel data access in zones that have the highest potential aggregate bandwidth.

A. Architecture

Figure 4 depicts the ZoneFS architecture, which organizes data center compute and storage resources into one or more *zones*, which consist of a leaf switch and its attached set of compute and storage servers. The number of zones can grow with the number of leaf switches in the data center, enabling vast and incremental expansion of the data center. In addition, ZoneFS uses a zone mapping manager to maintain a mapping of the files and directories stored within each zone.

Limiting the striping of a file to a switch and its connected servers avoids many of the network oversubscription pitfalls discussed earlier. For example, stand-alone applications, Internet workloads, and moderate sized parallel applications can all experience the performance of a parallel file system without expensive switching hardware. In addition, ZoneFS allows zones to contain any number of servers, allowing administrators to create zones of varying sizes to handle different types of workloads.

1) Zones: A zone consists of a switch and its attached servers. Servers are logically divided into a set of compute servers and storage servers, with file data and metadata striped across the storage servers. Compute servers generate I/O requests to the storage servers on behalf of applications. Storage servers are attached to any number of storage mediums such as direct-attached disks, SSDs, or a SAN. This separation allows administrators to turn off under-utilized compute nodes for power conservation.

The logical division of storage and compute servers allows a great deal of flexibility in the composition of the zone. If every server has access to storage, then every server can serve data requests. On the other hand, all storage devices can be consolidated in a few robust storage servers that are outfitted with quad-GigE or 10 GigE cards and connected to commodity disk arrays. This allows compute intensive data centers to focus on a greater number of compute servers per zone and storage heavy data centers to increase the number of storage servers per zone. Another option would be to set up a small SAN for each zone, allowing direct data access from every server.

2) Zone Mapping Manager: For applications to take advantage of ZoneFS and avoid inter-switch bottlenecks, it is important that applications are launched on one or more servers in the correct zone. Moving compute to a zone is 20 to 40 times (depending on how many servers are in your zone) more flexible than the Internet scale file system method of moving compute to a particular server.

A set of *zone mapping managers* maintain a mapping of the files stored within each zone. A cluster job scheduler uses this service to launch applications on compute servers within the correct zone. Once an application is running in a zone, it accesses data on the local storage nodes.

ZoneFS scales metadata access by dividing the information into two separate groups: file metadata and data location information. File metadata is stored on storage servers within the zone in which the file is located. ZoneFS clients or a cluster

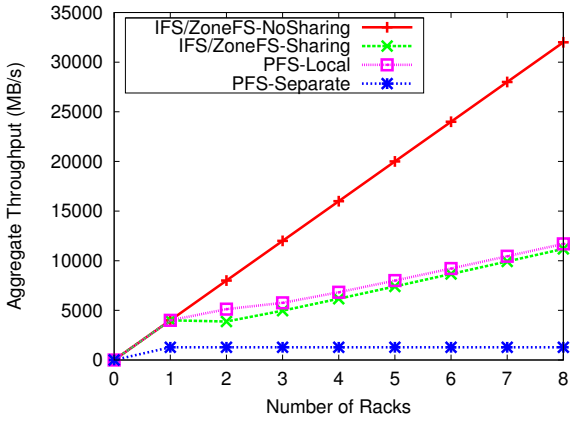

Fig. 5. **Data Center Scalability**. *ZoneFS and Internet file systems can scale throughput to data in separate racks, but all architectures are constrained when accessing data on different racks. This is clearly demonstrated as we increase the number of rack from 1 to 2. Each rack has 40 nodes. We simulated up to 48 racks but show up to only 8 racks for clarity. The scaling trend is the same for the remaining racks (until the upper-level switch runs out of ports).*

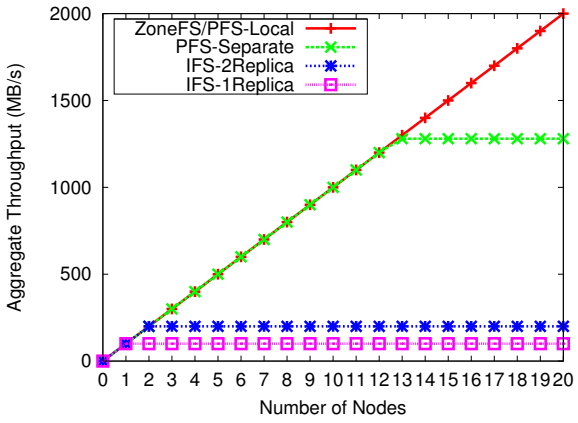

Fig. 6. **Single Rack Scalability**. *While Internet file systems can scale only with the number of replicas, ZoneFS and GPFS can scale with the number of nodes on which data is striped. Each rack has 40 nodes, but for clarity up to only 20 nodes are shown since the trend continues.*

job scheduler access and update data location information, i.e., the mapping of files to zones, by accessing a zone mapping manager.

Note that applications running in one zone can access data in any zone in the data center, but are subject to the available bandwidth between leaf switches. In addition, our current design places files within a single zone, but we are investigating whether we can stripe large chunks of a file in separate zones.

B. Scalability Simulations

Validating the design of ZoneFS in a large data center would be extremely beneficial, but any test infrastructure with fewer than 1000 nodes would fail to prove the benefits of ZoneFS beyond a reasonable doubt. Instead, we validate the design of ZoneFS by simulating the scalability of different file system architectures performing sequential I/O of a large file in a data center. Our data center uses a 40 nodes per rack, a GigE top of rack switch, a 10 GigE upper-level switch, and each node contains a disk subsystem that can sustain 100 MBps.

Figure 5 validates the scalability of ZoneFS as we increase the number of racks. For Internet file system (IFS) and ZoneFS, the NoSharing throughput represents workloads that do not transfer data between racks. For these workloads, aggregate throughput scales linearly with the number of racks in the data center, independent of the upper-level switches. The Sharing experiments represent workloads that do transfer data between racks, e.g., Reduce phase. For these workloads, the type of file system architecture is less important, as all architectures end up being constrained by the 10 GigE upper level switch. Finally, PFS-Separate represents a parallel file system that is connected to compute nodes through a dedicated network switch as described in Figure 2. As the number of racks increase, the bandwidth of this architecture is limited to 10 GigE by its dedicated switch.

For data access within a rack, Figure 6 shows the throughput as we increase the number of nodes reading a single chunk of data. With Internet file systems (IFS), the bandwidth to the data scales with the number of replicas (or network links) to the data. The PFS-Separate architecture does better, but is constrained by the upper-level 10 GigE switch. ZoneFS and PFS-Local are the only architectures that stripe the data across all the nodes and therefore scale with the number of nodes in the rack.

C. High Availability

While this paper focuses on the performance benefits of ZoneFS, it is worth discussing how ZoneFS can handle disk, node, and rack failures. While Internet-scale file systems rely on triplication (mainly for cost reasons), the flexible architecture of ZoneFS allows for several different alternatives depending on the storage subsystem architecture.

To recover from disk failures, each storage node can employ RAID across its local disks.

To recover from storage server failures, for SAN architectures, any storage server can serve any piece of data. Compute nodes and file system clients can simply reroute their requests to other storage nodes within the zone. For local disk architectures, data can be either replicated across a zone's storage servers, or save disk space by using client-driven (network) RAID 5/6 across the storage servers [9]. Either way, all storage nodes can participate in reconstruction, greatly decreasing time to recovery. Another option is to not install any disks within a storage server at all, but rather connect the storage servers to JBODs or SAS disk arrays. In this configuration, the disk array interface cables can be connected to multiple storage servers, allowing failover from one storage server to another.

To recover from the failure or planned shutdown of a zone (possibly due to a top of rack switch failure), one option is for ZoneFS clients and storage servers to replicate data between

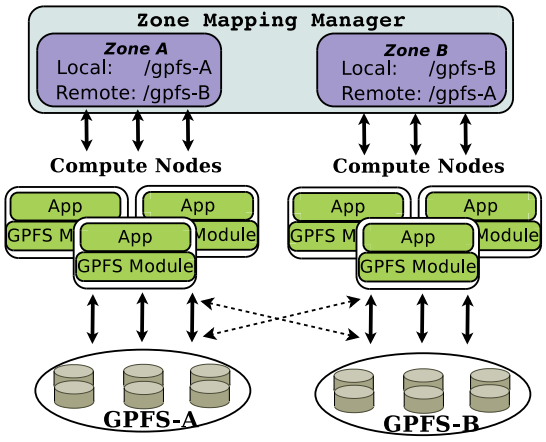

Fig. 7. **ZoneFS Prototype.** *Zone mapping manager keeps track of all files and directories stored in each zone. Compute nodes stripe data within a zone for local data access and mount remote zones for remote data access.*

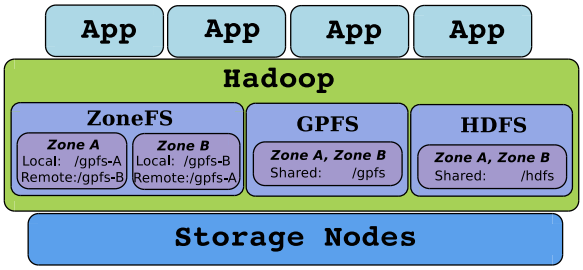

Fig. 8. **Hadoop-GPFS Plug-in Layer.** *This layer supports GPFS and ZoneFS as underlying file system in Hadoop by using Hadoop abstract file system API.*

zones and update the zone mapping manager with new location information. This technique would force data to pass through the oversubscribed upper level switches (just like replication in HDFS). One way to avoid this would be to connect JBODs or SAS disk array interface cables to storage servers in two zones. This allows storage nodes in one zone to handle I/O requests for data in another zone, even while their switch is powered down.

IV. ZoneFS Implementation

We implemented a ZoneFS prototype that is layered upon the IBM GPFS parallel file system. To support data-intensive analytic workloads, we also modified Hadoop to support GPFS and ZoneFS as primary file systems.

A. ZoneFS Prototype

As shown in Figure 7, we implemented a prototype of ZoneFS in a layer on top of a single running instance of GPFS for Linux. Storage nodes can support either local disk or a shared SAN per zone.

To perform I/O within each zone, compute nodes stripe data across only its local storage nodes. To perform direct and parallel I/O to the storage nodes in a different zone, compute nodes must "mount" the storage nodes from each remote zone. For example, in Figure 7, there are two clusters: cluster A and cluster B. To get a global view of all the files in the system, each zone mounts both GPFS-A and GPFS-B with different locality attributes. ZoneFS clients can access data in other zones by first contacting the remote zone metadata manager to determine the zone's storage nodes. The ZoneFS client can then use parallel I/O to access the remote file. While these clients are subject to the inter-switch bottleneck, they prevent storage hot spots by balancing requests across all available storage nodes.

The zone mapping manager maintains mappings of all files and directories stored in each zone, assigning a "local" or "remote" attribute to directories depending on the zone in which they are located. To assign jobs to specific zones, we implemented a job scheduler that works with both MPI and Hadoop. The job scheduler uses the zone mapping manager to assign jobs to zones based on the pathname prefix of the required data for the job. Within each zone, the specific compute nodes that are utilized depend on many factors, e.g., current number of running jobs, CPU utilization. Currently, our prototype evenly load balances jobs across all nodes. Our prototype maps directory trees to zones, which drastically reduces the number of application mapping requests.

Our implementation of ZoneFS is separate from the file system and is compatible with other parallel file systems that can be run on commodity hardware and utilize direct-attached disks or a SAN. It is interesting to note that ZoneFS does not require all zones to use the same file system across its storage nodes. Heterogeneous data centers that use multiple file system across their compute clusters can use ZoneFS to build a single namespace and allocate jobs across all available file systems.

B. Hadoop-GPFS Plug-in Layer

In order for Hadoop to support a file system, it must be able to determine all available storage nodes and their associated data blocks. To support GPFS and ZoneFS, we leveraged the Hadoop abstract file system API to implement Hadoop-GPFS and Hadoop-ZoneFS plug-ins. Hadoop uses these plug-ins to determine the optimal server to utilize for each data block.

Our modified Hadoop architecture is shown in Figure 8. Our GPFS and ZoneFS plug-ins intercept file system data block requests and determine the node on which to launch a map/reduce job. For GPFS, the plug-in returns a random node from any zones, which works because data blocks are striped across all storage nodes in the data center. For ZoneFS, the plug-in uses the zone mapping manager to return a random node in the zone that contains the requested data.

V. Evaluation

In this section, we evaluate ZoneFS, GPFS, and HDFS under typical data center workloads and cluster configurations. We first compare the I/O scalability and performance of GPFS and ZoneFS using the IOR microbenchmark. Next, we analyze file system performance with large I/O requests using two Hadoop applications. We then use both a microbenchmark

and a Hadoop application to investigate the ability of these file systems to provide predictable performance regardless of how data is laid out across the storage nodes. Finally, we analyze small and medium sized I/O request performance with an image conversion cloud computing application.

Note that ZoneFS targets oversubscribed hierarchical data center network architectures, and as such we see supporting HPC applications as future work.

A. Experimental Setup

In our experiments, we compare ZoneFS with both standard GPFS and the Hadoop DFS. In addition, with GPFS and ZoneFS we use both a SAN, which is used by many parallel file systems, and a local disk storage configuration, which is used by many data centers.

Our experiments compare the following configurations:

- **HDFS**: Hadoop DFS with local disks
- **GPFS-Local**: Standard GPFS with local disks
- **GPFS-SAN**: Standard GPFS with SAN
- **ZoneFS-Local**: Switch-Aware GPFS with local disks
- **ZoneFS-SAN**: Switch-Aware GPFS with SAN

We use Hadoop 0.20.0 with block size of 64MB. Both GPFS and ZoneFS use a stripe size of 1MB. Write experiments are complete when all data is on disk. Read experiments use an empty data cache.

All experiments are conducted on a 16-node cluster, with 8 nodes on each of 2 racks. Three Netgear gigabit switches are used to connect nodes within each rack and the two racks together. A GigE link connects the leaf switches to the parent switch. Each node is equipped with dual 3 GHz Xeon processors, 4 GB memory, and a local disk that runs Red Hat Enterprise Linux 5.2. All nodes have access to a shared Fibre Channel SAN, which is comprised of a 16-port FC switch connected to an IBM DS4700 storage controller. Each node has 5 hard drives using RAID5. For local disk configurations, nodes can access data on their local disks directly, but must communicate with other nodes for access to their data. For SAN configurations, each rack is allocated a set of devices and cannot directly access devices on the other rack, which must be done through the upper-layer switch.

B. Scalability: GPFS VS ZoneFS

This section compares scalability of standard GPFS, which stripes files across every node on both racks, and ZoneFS, which stripes files only within a zone. We use the IOR 2.10.1 benchmark [15] to read and write separate 2GB files.

1) Local and Cross Rack Performance: We first motivate the design of ZoneFS by comparing the performance of accessing files solely within a single rack vs. across racks. As expected, Figure 9 demonstrates that striping files across multiple racks (Read Remote and Write Remote) bounds I/O performance by the upper level gigabit switch—increasing the number of nodes does not increase the read or write I/O throughput. GPFS-SAN (Read Local) saturates the available storage bandwidth with only 2 nodes, demonstrating that a better SAN could improve performance even further. For write performance, GPFS-SAN (Write Local) saturates the available bandwidth of the storage controller, achieving more than double the cross rack write performance.

2) Cluster Performance: Now that we understand the baseline inter- and intra-rack I/O performance of GPFS, we compare ZoneFS and GPFS on the whole cluster. With ZoneFS striping data within each individual rack, Figure 10 shows that ZoneFS-SAN can continue to saturate the available storage bandwidth. GPFS-SAN reads and writes data from both the local and remote rack. As such, it can perform fast I/O within the rack, but is limited by the inter-switch network bandwidth for data blocks on the remote rack. Consequently, its performance is between the baseline inter-rack and intra-rack values.

C. Analytics Workloads

As data-intensive applications are growing in popularity, any file system for the data center must excel for these workloads. Our goal is to demonstrate that ZoneFS can outperform GPFS and match the runtime performance of HDFS, the file system built specifically for Hadoop. We choose two representative Hadoop workloads: Teragen and Terasort. Each experiment uses weak scaling with each active node accessing a separate 1 GB file.

For these experiments, we re-create the 10:1 network oversubscription bottlenecks in large data centers by scaling down the entire system, using only 8 nodes per rack, a GigE rack switch, and a 100Mbps inter-switch network link [7].

We find that ZoneFS can leverage switch-awareness to achieve similar performance as HDFS, which uses node awareness. In addition, these experiments demonstrate that the specialization of HDFS, while possibly simplifying its implementation, seems to limit the range of applications HDFS can support and not increase its performance. On the other hand, GPFS, which stripes data across multiple racks, has poor performance due to the inter-switch bottleneck.

1) Teragen: Teragen is a write-intensive workload that generates the input data sets for Terasort. Each data set contains of a series of records, each consisting of a key, row id, and filler.

Figure 11 shows the execution time of GPFS, ZoneFS, and HDFS with the Teragen workload. With 16 nodes, GPFS-SAN's execution time is 1.70 times that of ZoneFS-SAN and 1.55 times that of HDFS respectively. ZoneFS and HDFS performance is similar, as both can saturate the storage subsystem. For ZoneFS, using a SAN improves performance by 8.5% over HDFS or ZoneFS-local disk since it can more evenly balance write requests across all available storage devices.

2) Terasort: Terasort sorts the output data sets from Teragen. It samples the keys of input data to get the partition points, then launches jobs to perform a parallel merge sort. Terasort is initially read-intensive as nodes read their input data, but becomes write-intensive as nodes write intermediate and merged results.

Terasort results are shown in Figure 12. Similar to the Teragen results, GPFS incurs the inter-switch network bottle-

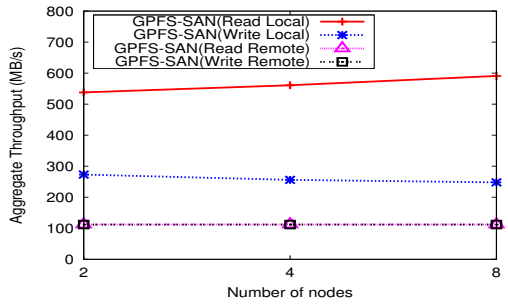

Fig. 9. **Cross-rack and Single-rack I/O performance.** *By striping files across multiple racks, I/O performance is limited by the gigabit inter-rack link. By striping files within local SAN, GPFS-SAN can scale with available SAN bandwidth.*

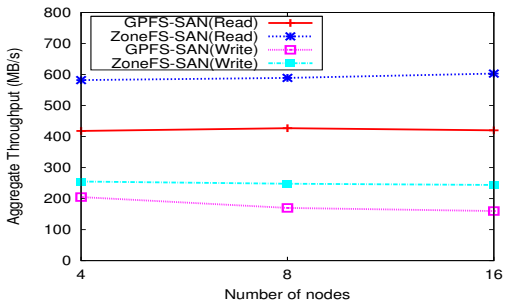

Fig. 10. **Multi-rack I/O performance.** *Since ZoneFS stripes data within each rack separately, it avoids the inter-switch bottleneck and saturates the underlying storage subsystem.*

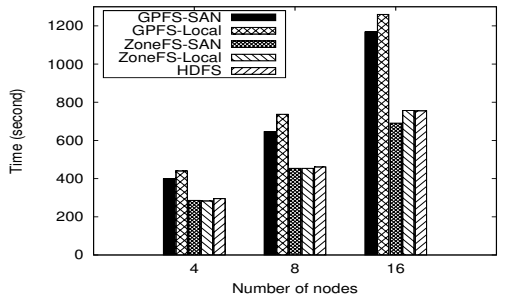

Fig. 11. **Hadoop Teragen write-intensive application.** *ZoneFS and HDFS performance are similar as the number of nodes increases. GPFS execution time increases as the number of nodes increases due to striping across multiple racks.*

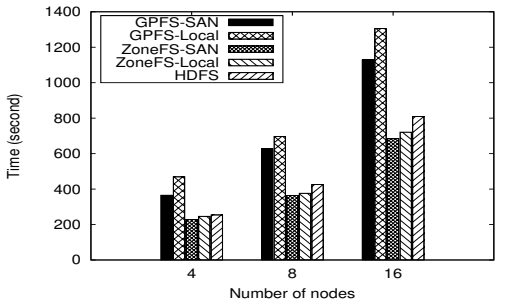

Fig. 12. **Hadoop Terasort application.** *Terasort exchanges a lot of data between rack to merge sorted data sets, which further constrains the available inter-rack bandwidth as the number of nodes increases.*

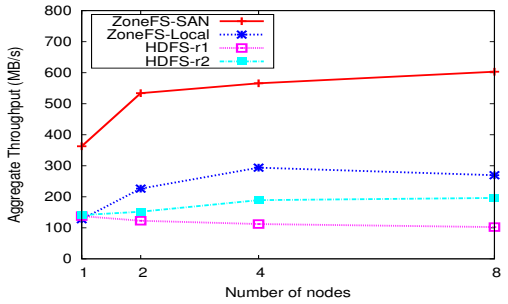

Fig. 13. **Hotspot read performance.** *When all data is located on a single data node, HDFS-r1 hits a single server bottleneck as the number of nodes increases. With replication, HDFS-r2 increases performance proportionally to the number of data copies.*

neck for both read and write data access. The Terasort shuffle phase generates inter-switch data transfers, which increases the runtimes of both ZoneFS and HDFS as the number of nodes increases. ZoneFS narrowly outperforms HDFS with either a SAN or local disk.

D. Data Hotspots

The primary advantage of ZoneFS is data striping in hierarchical network architectures, which allows independent applications running within one or more zones in a data center to have their I/O requests balanced across all the disks on all servers within a zone. Fine-grained data striping prevents *hotspots* from occurring by equally distributing I/O requests across multiple servers and their associated disks. With HDFS, applications write data to the local disk, which is accessible via a single node. This node can consequently become a hotspot if many tasks need to access the data on that node. This section analyzes the effect of these hotspots on I/O throughput and application run time and if replication can help to mitigate the problem.

This experiment has each node read a separate 5 GB file from nodes within a single rack. These files were all created in ZoneFS and HDFS from a single node. HDFS-r1 uses a replication factor of one, which means that a single copy of the data is stored in the file system. As shown in Figure 13, HDFS-r1 quickly hits a bottleneck in accessing the single node that contains the dataset. Many data centers replicate the data on two nodes within a single rack, which is represented by HDFS-r2. We can see that while replication mitigates the hotspot, it does so by only a factor relative to the number of replicas. With ZoneFS-Local, even though data is evenly striped across all the storage nodes, it is still constrained by incast in the gigabit switch. ZoneFS-SAN uses a much better quality FC switch and can evenly distribute I/O requests directly across all available disks, and achieve the maximum throughput of

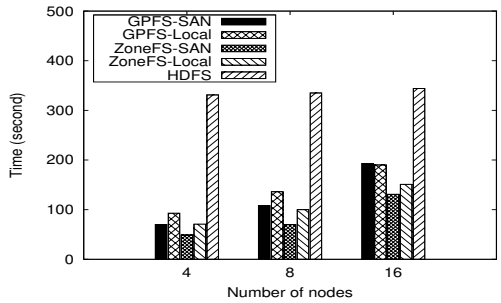

Fig. 14. **Image conversion workload: 1 MB JPG to 9 MB BMP.** *Elapsed time to convert 200 images per node. ZoneFS and GPFS use their distributed architecture to achieve a low runtime. HDFS runtime is much longer due to its inefficient handling of small I/O requests.*

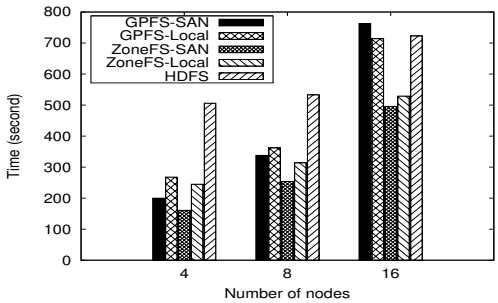

Fig. 15. **Image conversion workload: 4 MB JPG to 35 MB BMP.** *Elapsed time to convert 200 images per node. ZoneFS runtime continues to outperform HDFS and GPFS. GPFS runtime increases substantially as the number of nodes increases, which saturates the inter-rack network bottleneck.*

the cluster.

E. General Applications

With the increasing popularity of cloud computing services such as Amazon EC2, successfully handling traditional applications is critical. A well known example is the New York Times, which used 100 EC2 virtual machines instances to convert 11 million articles to PDF format [16]. In this section, we analyze I/O performance of applications that generate small and random I/O requests across the file system. This I/O workload is typical of many different applications including OLTP, virtual machine (VM) accesses to virtual LUNs, e.g., VMware .vmdk file, among many others.

To represent this workload, we use an image conversion benchmark to convert the encoding of 200 images (per node) from JPG to BMP using the Python Imaging Library. Image conversion, which consists of sequential and random reads and writes to many small to medium files, is similar to the workload exhibited by OLTP and mail server applications. In addition, beyond the well known New York Times example, image conversion is a very common operation for data centers that allow users to upload images and movies, e.g., Facebook, Google Picasa, since they are not stored in the format in which they were uploaded.

We use two racks, each with 2 to 8 nodes and a GigE switch, and use a single 1Gbps link between the two racks. With HDFS, we also ensure that the file conversion application executes on the same node as its input images. Even though HDFS has specialized architecture and was not designed to support large numbers of files with a size less than 128 MBs, these experiments demonstrate a very important point. While the Hadoop experiments showed that the general local disk (and SAN) architecture of ZoneFS could match the specialized architecture of HDFS, these cloud workload experiments demonstrate the degree to which this specialization has hindered HDFS for general applications.

In Figure 14 we show the results when converting a 1 MB JPG to a 9 MB BMP file. HDFS uses a single NameNode that stores the server location of blocks in the file system. With even the moderate number of small files, the HDFS NameNode is overwhelmed with location requests and takes two to six times as long to complete the benchmark. GPFS, on the other hand, has a fully distributed architecture that scales with the number of metadata requests, allowing it to outperform HDFS. ZoneFS can leverage this distributed architecture and outperform GPFS by avoiding the inter-switch bottleneck. As we increase the number of nodes, the execution time of ZoneFS and GPFS increases as nodes compete for access to the storage subsystem. In addition, GPFS execution time increases even further as the increasing number of nodes start to saturate the inter-switch link.

In Figure 15 we show the results when converting a 4 MB JPG to a 35 MB BMP file. ZoneFS continues to have the lowest runtime, running 64% faster than HDFS with 4 nodes. With HDFS, since all file sizes are still less than the 64 MB block size, the number of location requests to the NameNode has not increased from the previous experiment. Consequently, HDFS runtime actually improves with the larger dataset. This demonstrates that HDFS is simply more efficient at accessing larger chunks of data, even if the data is local to the application. For GPFS, as the number of nodes increases, the inter-switch bottleneck becomes a limiting factor, and its runtime exceeds that of even HDFS.

F. Discussions

1) Scalability: ZoneFS does not limit the number of racks and switches that it can support. As the number of nodes per rack increase, the available disk and network bandwidth will increase proportionately. The ability to stripe data across all nodes within a rack enables it to scale with available storage and network bandwidth. This allows ZoneFS to avoid single node and their associated disks to become bottlenecks.

2) Efficiency: Our experiments show that the local disk architecture performed only slightly worse than the SAN-per-rack architecture for Hadoop workloads. This is because the efficiency of Hadoop is relatively low [17], which means that Hadoop cannot drive the I/O subsystem of each individual node. For Teragen and Terasort, the maximal I/O throughput per node during execution is 40 MB/s, which is much less than the peak IO throughput of a storage node. As the efficiency

of Hadoop improves over time, ZoneFS will be able to ride the performance wave of improvements. In addition, with commodity 10 Gbps switches emerging on the market, the performance of the local disk architecture will improve if not exceed that of a low end SAN such as the one we used for our experiments.

3) Flexibility: We used a SAN architecture to determine the potential benefits of using a more expensive solution than local disk. Interestingly, beyond the standard benefits of a SAN, e.g., storage virtualization, we found little performance benefits. In addition, although all of our directly-attached disk experiments used the same nodes for both compute and storage, we believe that separating compute and storage nodes can bring similar high-availability benefits as a more expensive SAN solution. ZoneFS clients can stripe across robust storage nodes that are outfitted with more disk and network bandwidth (possibly using client driven or network RAID). In addition, the decoupling of storage and compute nodes would allow better flexibility to turn off under-utilized compute nodes for power conservation.

VI. Related Work

Parallel file systems have been widely deployed in high performance computing environments, including IBM GPFS [8], Lustre [10], PVFS [18] and Panasas [9], all of which rely on massive aggregate bandwidth between separate compute and storage nodes. Several of these file systems allow users to specify per-file striping parameters, e.g., stripe size, RAID number, but they continue to lack knowledge of the data center network topology.

On the other end of the spectrum, Internet-scale file systems [6], [5], [19] and even some HPC targeted file systems [20] all try to avoid the network altogether by attempting to co-locate compute processing with large chunks of data on individual nodes. This swing away from parallel I/O has many roots, including the existence of incast, the lack of an affordable 10 GigE switching fabric, and an understanding of how to limit striping to avoid oversubscribed switches. Today, incast is an understood phenomenon that can be avoided (to some extent) [21], 10 GigE switches are 50% cheaper that GigE switches when considering per-MB/s throughput cost, and file systems like ZoneFS give insights into how to control file striping.

Investigation has begun into understanding Hadoop-specific enhancements to the PVFS [22] and GPFS [23] parallel file systems. These efforts focus on co-locating compute and data on a single node, i.e., making parallel file systems act more like Internet scale file systems. By writing large chunks of data to local disk, these enhancements mitigate key benefits of parallel file systems, load balancing across all available servers and the ability to avoid data loss without having to resort to inter-node replication.

Recent work in data-center network design strives to eliminate oversubscription to the core switching fabric [24], [25], [26]. This work focuses on removing the inter-switch bottleneck and is therefore complementary to ZoneFS, which focuses on improving I/O performance within a single switch. Some of this research appears as though it could enable wide-striping of data across all racks in a data center, but this continues to have several drawbacks. First, while incast [27], [28] can be mitigated, every additional switch between the source and target presents an opportunity for buffer overflow, and second, wide striping of data means that performance will be limited to the slowest node on which the data resides, which means that striping too wide may start to reduce performance.

Job placement tools such as Tashi [29] and Dynamo [30] seek to optimize the co-location of compute and data through an understanding of the data placement within the data center. To assign jobs across the data center, Tashi uses a fine-grained Data-Location service to track data blocks and their associated server. ZoneFS is a candidate file system for use with Tashi. In fact, ZoneFS simplifies and possibly even increases the scalability of Tashi's Data-Location service by mapping data to an entire zone.

Porter explores the advantages and limitations of decoupling storage from computation in Hadoop [31]. Each SuperDataNode (SDN) contains an order of magnitude more disks than traditional Hadoop nodes. This proposal does not support parallel file access across multiple SDNs, thus each SDN must be managed in isolation without a single complete file system management framework. Consequently, each SDN can become an I/O bottleneck for "hot" data. ZoneFS avoids these bottlenecks by using parallel I/O across all storage nodes within a zone.

VII. Future Work

ZoneFS targets oversubscribed hierarchical data center network architectures, not supercomputer architectures. As such, ZoneFS does not target HPC workloads since MPI applications that share ghost cells on every iteration would perform unacceptably slow in an oversubscribed data center. But ZoneFS does support efficient execution of HPC workloads within a single zone, and it would be a very interesting to investigate whether such HPC applications could be modified to account for inter-switch bottlenecks.

We are currently implementing ZoneFS in a layer above the file system. This allows our prototype to work with any parallel file system, but at the price of additional management and setup overhead. Integrating directly with GPFS (or any other parallel file system) would not only remove this overhead but would also provide more transparency to administrators.

Exploring local-disk and SAN storage systems with a wider variety of applications workloads such as web-servers and databases would allow us to provide more insight into their true costs and benefits. For example, with local-disk architectures, nodes have a higher load since they must both issue and serve data requests. This was not a factor in our experiments, but may become an issue with other workloads.

ZoneFS provides flexibility in recovering from failures. To avoid replication within a zone, we would like to explore the use of client-driven (network) RAID 5/6 and/or declustered RAID across the storage servers [9].

VIII. Conclusions

This paper introduced ZoneFS, a parallel file system that optimizes the use of data striping and parallel I/O with hierarchical network architectures that are common in modern data centers. ZoneFS combines the scalability, performance, ease-of-management, and standard semantics of a parallel file system with the cost-effectiveness of an Internet-scale file system. By understanding the data center network topology and maintaining the logical separation of storage and compute nodes, ZoneFS provides an extremely flexible architecture that can avoid network bottlenecks and saturate available bandwidth.

We evaluated our prototype of ZoneFS using several data and metadata micro-benchmarks, as well as several applications that are mission critical for cloud infrastructures. ZoneFS data striping allows independent applications running within one or more zones in a data center to have their I/O requests balanced across all the disks on all servers within a zone. For Hadoop applications that operate on large data sets, we found that ZoneFS could avoid server hotspots and outperform GPFS and match or outperform HDFS with identical hardware and storage configurations. We also demonstrated that ZoneFS could scale linearly with a general cloud workload with a range of file sizes.

References

[1] Nature, "Big Data," 2008, http://www.nature.com/news/specials/bigdata.

[2] Hadoop, "Apache Hadoop," in *http://hadoop.apache.org/*, 2010.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[4] L. A. Barroso and U. Holzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool, 2009.

[5] HDFS, "HDFS: Architecture and Design," in *http://hadoop.apache.org/hdfs/*, 2010.

[6] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[7] Cisco, "Cisco Data Center Infrastructure 2.5 Design Guide," 2007, http://www.cisco.com/univercd/cc/td/doc/solution/dcidg21.pdf.

[8] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, 2002.

[9] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable Performance of the Panasas Parallel File System," in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, 2008.

[10] Sun, "LUSTRE FILE SYSTEM: High-Performance Storage Architecture and Scalable Cluster File System," in *White Paper*, 2008.

[11] S. Oehme, J. Deicke, J.-P. Akelbein, R. Sahlberg, A. Tridgell, and R. L. Haskin, "IBM Scale Out File Services: Reinventing Network-attached Storage," in *IBM Journal of Research and Development. VOL. 52 NO. 4/5*, 2008.

[12] M. Eisler, P. Corbett, M. Kazar, D. Nydick, and C. Wagner, "Data ONTAP GX: A Scalable Storage Cluster," in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, 2007.

[13] BlueArc, "Bluearc," 2010, http://www.bluearc.com.

[14] "Specsfs2008," http://www.spec.org/sfs2008.

[15] IOR, "IOR HPC Benchmark," 2009, http://sourceforge.net/projects/ior-sio/.

[16] D. Gottfrid, "Self-service, Prorated Super Computing Fun!" 2007, New York Times Blog.

[17] E. Anderson and J. Tucek, "Efficiency Matters!" in *Proceedings of Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2009.

[18] W.B. Ligon III and R. Ross, "PVFS: Parallel Virtual File System," in *Beowulf Cluster Computing with Linux*. MIT Press, 2001.

[19] KFS, "CloudStore (formerly, Kosmos File System)," 2010, http://kosmosfs.sourceforge.net/.

[20] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi, "Grid Datafarm Architecture for Petascale Data Intensive Computing," in *Proceedings of IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, 2002.

[21] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *Proceedings of ACM SIGCOMM*, 2009.

[22] W. Tantisiriroj, S. Patil, and G. Gibson, "Crossing the Chasm: Sneaking a Parallel File System into Hadoop," in *Proceedings of Petascale Data Storage Workshop*, 2008.

[23] R. Ananthanarayanan, K. Gupta, P. Pandey, H. Pucha, P. Sarkar, M. Shah, and R. Tewari, "Cloud Analytics: Do We Really Need to Reinvent the Storage Stack?" in *Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud),*, 2009.

[24] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM)*, 2008.

[25] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM)*, 2009.

[26] EDC, "Ethernet in the data center," 2010, http://www.ethernetalliance.org.

[27] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems," in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, 2008.

[28] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC)*, 2004.

[29] M. A. Kozuch, M. P. Ryan, R. Gass, S. W. Schlosser, D. O'Hallaron, J. Cipar, E. Krevat, J. López, M. Stroucken, and G. R. Ganger, "Tashi: Location-aware Cluster Management," in *Proceedings of Workshop on Automated Control for Datacenters and Clouds*, 2009.

[30] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," in *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, 2007.

[31] G. Porter, "Decoupling Storage and Computation in Hadoop with SuperDataNodes," in *Proceedings of ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2009.