The Panasas ActiveScale Storage Cluster – Delivering Scalable High Bandwidth Storage

David Nagle, Denis Serenyi, Abbie Matthews

Panasas Inc.
Fremont, CA

{dnagle, dserenyi, amatthews}@panasas.com

Abstract

Fundamental advances in high-level storage architectures and low-level storage-device interfaces greatly improve the performance and scalability of storage systems. Specifically, the decoupling of storage control (i.e., file system policy) from datapath operations (i.e., read, write) allows client applications to leverage the readily available bandwidth of storage devices while continuing to rely on the rich semantics of today's file systems. Further, the evolution of storage interfaces from block-based devices with no protection to object-based devices with per-command access control enables storage to become secure, first-class IP-based network citizens. This paper examines how the Panasas ActiveScale Storage Cluster leverages distributed storage and object-based devices to achieve linear scalability of storage bandwidth. Specifically, we focus on implementation issues with our Object-based Storage Device, aggregation algorithms across the collection of OSDs, and the close coupling of networking and storage to achieve scalability.

1. INTRODUCTION

The last few years have seen a fundamental shift in storage architectures. Specifically, the monolithic, central-service architecture has given way to the distributed storage system. Key to the scalability of distributed storage has been the separation of computationally expensive metadata management from the bandwidth intensive data path. This separation allows clients (or servers) to directly access storage by providing them with metadata layout information. Clients use the metadata to directly read and write storage devices, avoiding expensive data copies through the metadata manager and delivering scalable bandwidth. Numerous systems have been built using this architecture, including EMC High Road, IBM Storage Tank, Carnegie Mellon University's (CMU) Network-attached Secure Disk (NASD), Sistina, the Lustre File System, and the Panasas Storage Cluster [EMC, Gibson98, Pease02, Sistina, Lustre, Panasas].

Another key change still underway is the advancement of storage from a block-based interface (e.g., SCSI, ATA) to an Object-based storage device (OSD) interface such as those used in Lustre, EMC Centera, and the Panasas Storage Cluster [Azagury02, EMC, Gibson98, Lustre03]. Currently in letter ballot at the SCSI T10 committee, the OSD V1.0 Standard defines an interface that allows storage devices (e.g.,. disks, tape, RAID arrays), to manage the layout of object data within the device, where intimate knowledge of data layout, head scheduling, prefetching algorithms and cache management enables significant improvements in individual device performance [OSD].

The coupling of OSD with distributed storage is allowing storage systems to reach new levels of scalability. This was the promise proposed in CMU's NASD project and now demonstrated by both the Panasas and Lustre Storage Systems, with both currently delivering over 10 GBytes/sec. of storage bandwidth (aggregated over 100's of clients) and with significant room for scaling[1]. While the architecture of OSD and distributed storage has been essential to both systems' success, several key design and implementation issues also played a significant role in the success of achieving scalable bandwidth. In particular, specific implementation issues within the OSD, including layout policies and prefetching/cache allocation were essential in ensuring that OSDs could support a large number of clients. Also, OSD aggregation required careful integration of networking and striping algorithms to avoid network congestion that could overwhelm either the clients or OSDs.

In this paper, we outline some of the most important performance issues that we have encountered, and the solutions that Panasas has developed. We begin by describing the Panasas architecture, including an overview of our Object-based Storage Device

Proceedings of the ACM/IEEE SC2004 Conference, November 6-12, 2004, Pittsburgh, PA, USA.

0-7695-2153-3/04 $20.00 (c)2004 IEEE

1. The current limits on achieving higher bandwidths have been the cost of building the testbed of clients, network, and storage to test larger configurations.

(OSD), the partitioning of file system functionality between clients and manager, and our use of RAID for striping data across OSDs. We then discuss the performance-critical aspects of the OSD implementation, focusing on the OSD layout, prefetching and cache management algorithms. Next, we show how the Panasas Storage Cluster delivers bandwidth scalability across a range of clients. Finally, we examine the performance implications of this architecture on networking and how we tuned our storage system to maximize the networking performance while running over TCP/IP and Ethernet.

2. The Panasas Storage Architecture

At the heart of the Panasas ActiveScale Storage Cluster is a decoupling of the datapath (i.e., read, write) from the control path (i.e., metadata). This separation provides a method for allowing clients direct- and parallel-access to the storage devices, providing high bandwidth to both individual clients and to workstation clusters. It also distributes the system metadata allowing shared file access without a central bottleneck.

Panasas storage devices are network-attached OSDs. Each OSD contains two Serial-ATA drives and sufficient intelligence to manage the data that is locally stored. Metadata is managed in a metadata server, a computing node separate from the OSDs but residing on the same physical network.

Clients communicate directly with the OSD to access the stored data. Because the OSD manages all low-level layout issues, there is no need for a file server to intermediate every data transaction. To provide data redundancy and increase I/O throughput, the Panasas ActiveScale File System (PanFS) stripes the data of one file system across a number of objects (one per OSD). When PanFS stripes a file across 10 OSDs attached to a Gigabit Ethernet (GE) network, PanFS can deliver up to 400 MB/sec split among the clients accessing the file.

2.1 Details: Five Components

There are five major components in the Panasas Storage Cluster.

- The primary component is the object, which contains the data and enough additional information to allow the data to be autonomous and self-managing.
- The Object-based Storage Device (OSD), which is a more intelligent evolution of today's disk drive that can lay out, manage, and serve objects.
- The Panasas File System (PanFS) client module, which integrates into the client, accepting POSIX file system commands and data from the operating system, addresses the OSDs directly and stripes the objects across multiple OSDs.
- The PanFS Metadata Server (MDS), which intermediates amongst multiple clients in the environment, allowing them to share data while maintaining cache consistency on all nodes.
- The Gigabit Ethernet network fabric that ties the clients to the OSDs and Metadata Servers.

2.2 Objects

The object is the fundamental unit of data storage in the system. Unlike files or blocks, which are used as the basic components in conventional storage systems, an object is a combination of "file data" plus a set of attributes that define various aspects of the data. These attributes can define on a per file basis the RAID parameters (i.e., RAID level, stripe unit size, stripe size), data layouts, and quality of service. Unlike conventional block storage where the storage system must track all of the attributes for each block in the system, the object maintains its own attributes to communicate to the storage system how to manage this particular piece of data. This simplifies the task of the storage system and increases its flexibility by distributing the management of the data with the data itself.

Within the storage device, all objects are addressed via a 96-bit object ID. OSD commands access objects based on the object ID, the beginning of the range of bytes inside the object and the length of the byte range that is of interest (<object, offset, length>).

There are three different types of objects. The "Root" object on the storage device identifies the storage device and various attributes of the device itself, including its total size and available capacity. A "Group" object is a collection of the objects on the storage device that share resource management policies such as capacity quota. A "User" object carries the actual application data to be stored.

Associated with each object is a set of attributes, organized as pages with 2^{32} attributes per page and 2^{32} attribute pages. Some attributes, such as Last_Modify_Time and Physical_Capacity_Used, are defined and maintained by the OSD to ensure correctness and data integrity. Other attributes, such as File_Owner and the Virtual_File_Size are maintained by the PanFS file system.

2.3 Object-based Storage Device

The Object-based Storage Device contains two SATA disk drives, a processor, RAM and a Gigabit Ethernet network interface, allowing it to manage the local object store and autonomously serve and store data from the network. Inside the OSD, data is striped RAID 0 across the two SATA disks.

The Panasas OSD command set processes commands against objects (e.g., read(object 0x1234, offset=0x00, length=2 MB)), eliminating the need for clients to obtain, cache, and manage the fine-grained layout information common to block-based distributed storage systems. Because the PanFS client supports RAID

across individual OSDs, objects that constitute a file can be directly addressed in parallel, without an intervening RAID controller and enabling extremely high aggregate data throughput rates.

The OSD provides three major functions for the storage architecture:

- Data storage – Like any conventional storage device, OSDs manage the object data as it is laid out into standard tracks and sectors on the physical media. No block access is provided outside the OSD. Instead, clients request data using an object ID, an offset to start reading or writing data within that object and the length of the data region requested.

- Intelligent layout – The OSD uses its memory and processor to optimize the data layout and the pre-fetching of data from the media. For example, the object metadata provides the length of data to be written, allowing a contiguous set of tracks to be selected. Using a write-behind cache, a large amount of the write data is written in a small number of efficient passes across the disk platter. Similarly the OSD can do intelligent read-ahead or pre-fetching of an object, allowing for efficient head scheduling to maximize overall disk performance. To ensure data integrity, the OSD cache is battery backed, allowing the OSD to commit all dirty cached data to disk during a power-failure.

- Per object metadata management – The OSD manages the metadata associated with the objects that it stores. This metadata is similar to conventional inode data, including an object's data blocks and object length. In a traditional system, this data is managed by the file server (for NAS) or by the host operating system (for direct-attached or SAN). The Object Storage Architecture distributes the work of managing the majority of the metadata in the storage system to the OSDs and lowers the overhead on the clients.

The OSD reduces the metadata management burden on the Metadata Server (MDS) by maintaining one component object per OSD, regardless of how much data that component object contains. Unlike traditional systems where the MDS must track each block in every stripe on every drive, successive object stripe units are simply added to the initial component object. The component objects grow in size, but for each object in the system, the MDS continues to track only one component object per OSD reducing the burden on the MDS, and increasing the MDS's scalability.

2.4 PanFS – The Panasas File System

In order for the clients to read and write objects directly to each OSD, a PanFS kernel-loadable module is installed on the client. Within this module, PanFS provides four key functions.

- POSIX file system interface – PanFS provides a POSIX interface to the application layer which allows the application to perform standard file system operations such as

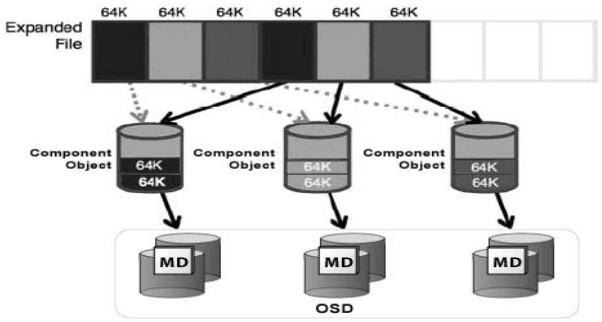

Figure 1: PanFS File to Object Layout. File data is striped across multiple objects, with one object per OSD.

`open()`, `close()`, `read()` and `write()` to the underlying storage system.

- Caching – PanFS provides caching in the compute node for incoming data complementing the cache in the OSD. There is also write-data caching that aggregates multiple writes for efficient transmission and data layout at the OSDs. A third cache for metadata and security tokens allows clients to quickly generate secure commands to access data on the OSDs for which they have been given permission. The client also supports client-based prefetching by tracking each application's requests and scaling prefetch requests from 64K to 4 MB based on the sequential access patterns of the application. In our Linux PanFS client, all data is cached using the Linux buffer/page caches.

- Striping/RAID – PanFS stripes files across multiple OSDs on a per file basis (Figure 1). Unlike standard RAID arrays, PanFS can apply a different data layout and RAID level to each file. PanFS takes a file and breaks it down to stripe units, which are the largest contiguous region of a file that is assigned to one OSD; that is, not split over two OSDs. The size of each stripe unit is specified as an attribute of the object (default is 64 KB) along with the RAID level (0, 1, or 5) and the number of OSDs that the object is striped across (data stripe width). If RAID 5 is specified, the number of objects used in each parity equation (parity stripe width) is also specified, and one stripe unit in each parity stripe will be used to contain the parity of the others, its value calculated and written by the client writing that parity stripe. The number of OSDs a file is striped across determines a file's maximum achievable bandwidth. Therefore, PanFS stripes the data of large files across all of the available OSDs.

- iSCSI – PanFS uses the iSCSI protocol to transmit the OSD SCSI command set and the data payload across our Ethernet/TCP network This use of standard protocols (i.e., TCP, iSCSI) allows us to leverage commodity hardware accelerators [vanMeter98,IETF].

2.5 Metadata Sever

The PanFS Metadata Server (MDS) controls the interaction of the clients with the objects on the OSDs, coordinating access to nodes that are properly authorized and maintaining cache consistency for users of the same file. The Metadata Server provides the following services for the Panasas ActiveScale Storage Cluster:

- Object storage access – The MDS constructs, manages, and disseminates a map describing the layout of each file, allowing clients to access objects directly. A map enumerates the OSDs over which the file is striped. The MDS also provides the client with a capability that describes and enables access to a file's component objects. A capability is a secure, cryptographic token used to demonstrate the client's authorization on each request to the associated object. OSDs verify the capability with each request they receive. The capability encodes the object ID over which the capability is valid, the associated privileges (e.g., read, write, create), and the length of time of capability is valid [Gobioff96, OSD]. To minimize communication between the client and MDS, the capability may be cached for its lifetime; the map is also cacheable. If a capability has expired or been invalidated by the MDS, the OSD will reject its use and the client will return to the MDS for an updated capability.

- RAID integrity – The MDS provides reconstruction of lost component objects and parity scrubbing for files that were open for write when a failure occurred.

- File and directory access management – The MDS implements a file structure on the storage system. This includes quota control, directory and file creation and deletion, and access control. In particular, when a client opens a file for the first time, the MDS performs the permission and access control checks associated with the file and returns the associated map.

- Client cache coherency – The MDS assists client-based file caching by notifying clients when changes in a cached file impact the correctness of client's cache (this is called a *callback*). When a client asks for `Read` or `Write` privileges to a file, a callback is registered with the MDS. When no other clients concurrently share a file, the requesting client is granted exclusive read or write access to the entire file and is likely to cache dirty data until the `close` system call and clean data through multiple open-close sessions. When all clients open a file for read-only, all clients are given callbacks to cache the entire file read-only. However, if client asks for a callback that is not compatible with a file's existing callbacks, then all clients that have callbacks are contacted, the appropriate write-back or cache invalidation is done, and new compatible callbacks are issued (including no callback or a non-cacheable callback).

2.6 Network Fabric

The Panasas Storage Cluster network provides the connectivity infrastructure that binds the Object-based Storage Devices, Metadata Server and clients into a single fabric, using TCP/IP over a Gigabit Ethernet for all communication. Because clients rely on TCP/IP, it is possible to use other networks such as Myrinet or InfiniBand. Hardware acceleration for link-level protocol conversion and routing is also available from some vendors [Myricom, Voltaire]. The Panasas Storage Cluster's key networking components are:

- iSCSI – The iSCSI protocol is used as the basic transport for commands and data to the OSDs, encapsulating SCSI inside of a TCP/IP packet. Currently, all requests from a client utilize the same iSCSI session to deliver the iSCSI Command Data Blocks (CDB) to the storage device and to read and write data.

- RPC command transport – PanFS has a separate lightweight Remote Procedure Call (RPC) that facilitates fast communication with the Metadata Server and clients.

- Other services – Many standard TCP/IP services are also needed to build Object Storage systems. For instance, NTP is used to synchronize storage system clocks. DNS is needed to simplify address management, and the various routing protocols allow the clients to be separated from the storage system. Such services are well established in the TCP/IP world and Panasas leverages their wide availability and inter operability.

3. Delivering Scalable Performance

3.1 Experimental Setup

All of the data in the following sections was taken using the Panasas Storage Cluster and 2.4 GHz P4 Linux clients (2.4.24), each with its own Gigabit Ethernet (GE) NIC. In the Panasas Storage Cluster, each OSD is connected to the network with a Gigabit Ethernet link. For wiring simplicity, Panasas provides a full crossbar GE switch that connects 10 OSDs internally to four aggregated (i.e., trunked) external GE links. In Panasas lingo, the 10 OSD configuration (plus one node for running the MDS), is called a "shelf" and is typically linked to the customer network switch via the 4-GE aggregated links. All of the experiments in the following sections use this configuration with one or more shelves, linked through an Extreme Black Diamond 6816 GE switch [Extreme]. For the scalability tests, each client sequentially reads (or writes) data from its own file. To ensure that the data is not pre-cached in the system, clients remount the file system before every test.

3.2 Implementing a High-Performance OSD

An efficient OSD implementation is essential to achieving scalable high bandwidth. Specifically, the Panasas OSD File System (OSDFS) strives to maximize head utilization for both read and write traffic patterns. OSDFS relies heavily on the object-interface, which exposes significantly more information to the storage device than block-based interfaces, allowing OSDFS to perform much more sophisticated management algorithms. OSDFS emphasizes efficient layout of each object, support for maximizing aggregate throughput under high load, intelligent prefetching algorithms that provide for efficient head scheduling, and cache management algorithms that balance cache utilization across many concurrent I/O streams.

To maximize head utilization on writes, OSDFS uses a write-behind algorithm to create large contiguous per-object disk layouts. As clients issue write commands to the OSD, OSDFS pulls data from the clients into the OSD battery-backed RAM cache, ensuring that data can be committed to disk even in the event of a power failure. When OSDFS's dirty data limit is reached (currently 32 MB), its block allocation algorithm sorts all dirty data for each object to ensure that objects are laid out contiguously to facilitate efficient reads of the data. Additionally, OSDFS attempts to write the entire 32MB of dirty data in a contiguous chunk of free disk space. If a 32 MB physically contiguous group of blocks is not available, OSDFS will break up the write into multiple seeks, always trying to place per-object data on physically contiguous blocks. Measurements of file layout show that even when 60 streams are concurrently writing to an OSD, an object's average contiguous region is 396 KB.

To maximize head utilization on reads, OSDFS's read-ahead algorithm aggressively uses information from the OSD interface to prefetch a significant amount of each active[1] object's data. The algorithm constructs a "read-ahead context" that it associates with a stream of sequential requests to an object. The read-ahead context stores the following information: 1) the address of the client stream's last read request, 2) a saturating counter that tracks how many times the context has been used successfully, and 3) last OSD read-ahead's address (max 6MB per stream). A single object may have many read-ahead contexts associated with it, each identified by the object-id and last-address read. This allows PanFS to support concurrent readers to the same object.

For every read, OSDFS checks to see if the request matches one of the read-ahead contexts for the specified object. A match is successful if the request sequentially advances the read-ahead context. OSDFS also allows forward skips of up to 1 MB to account for skipping over data parity within an object. If a match occurs, the OSD will advance the counters within that context, and may schedule a new disk fetch if the client's sequential stream is too close to the read-ahead stream. Typically, once a client has made several sequential accesses, the OSD prefetcher will have already fetched the data into the OSD cache.

It is important to note that Object-based storage provides a rich set of information to our read-ahead algorithm. By using object names, offsets and length-of-request (vs., block addresses and lengths), the interface allows us to very precisely control prefetching. The object model allows us to prefetch data on a logical basis, avoiding the useless prefetching of data from unaccessed objects that just happened to be physically contiguous to the current object(s) being accessed[2]. Furthermore, by designing the prefetch algorithm to track read-ahead contexts based on object-id and offset, instead of per-client information (e.g., client IP address or iSCSI session ID), OSDFS supports accurate read-ahead across multiple processes accessing the same object, even from the same client machine. If we had based our read-ahead algorithm on tracking requests on a per-client basis or limited ourselves to only one read-ahead context per object, then multiple processes accessing the same file would appear to generate a random request stream. By tracking multiple per-object read-ahead contexts, we successfully differentiate between different requests streams originating from the same physical machine.

Prefetching is closely tied to the OSD's dynamic cache management algorithm. With a small number of streams, each stream will read ahead a maximum of 6 MB, consuming a small amount of OSD cache for read-ahead data. As the number of streams grows, so does the amount of OSD cache consumed. OSDFS currently supports up to 150 MB of read-ahead cache, which means that 40 concurrent streams, each reading 6 MB/sec ahead would overwhelm the read-ahead cache. When this scenario actually happened during testing we quickly discovered two flaws in our algorithm. The first flaw was that our FIFO read-ahead cache eviction policy penalized slow clients. For example, if one client fetched its data more slowly than the other 40, it was highly likely that the one slow client's prefetched data would be evicted from the cache before it could be read. This miscalculation cost us twice. First, OSDFS had wasted disk time prefetching the data and then evicting it (for the slow client); second, when the slow client needed its data, the OSD had to perform a synchronous demand read, which reduces the effectiveness of OSD disk scheduling. This problem was quickly solved by changing our eviction policy to promote already consumed data ahead of non-consumed data.

The second flaw was an interaction between cache and read-ahead management algorithms. Instead of always reading ahead a maximum of 6 MB per stream, we reduced the amount of data each stream prefetched as the number of concurrent streams increased. This not only reduced the amount of cache used, but it reduced the amount of time any one stream would be prefetching data, providing a fair sharing of the disk among the N streams.

1. An "active object" is an object that has been referenced within a small window of time (currently 30 seconds).

2. Current OSD algorithms are not integrated inside the SATA drive's electronics. Therefore, the drive's own prefetching algorithms and buffer caches operate independently of the OSD prefetching and caching algorithms. Future generations of OSDs will benefit from a tighter coupling of drive electronics and OSD data management policies.

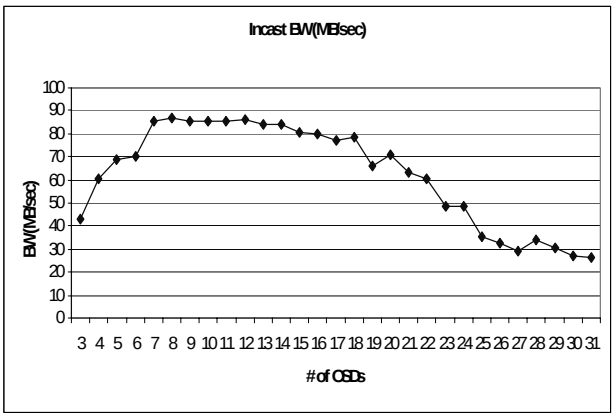

Figure 2: Linear Scalability and Incast Traffic Pattern. The left part of the graph shows almost linear scaling of bandwidth (for a single client) as the number of OSDs increases from 3 to 7. However, after 14 OSDs, the client's bandwidth actually drops. This behavior occurs because too many senders (OSDs) overflow the network buffers, causing significant packet loss We call this behavior incast. The graph's data was generated by a single client sequentially reading from a file using an 8 MB block size.

Therefore, we dynamically partition the read-ahead cache with N sequential streams each receiving (at most) 1/Nth of the cache space. We considered more sophisticated algorithms, where faster clients might be allowed to increase their read-ahead depth and cache space, but so far no client application has demonstrated this need. Our measurements show that the prefetcher can maintain over a 90% success rate, even with over 60 independent client streams.

4. Aggregation and Network Interactions

Key to achieving scalable bandwidth is aggregation by striping data across many OSDs. With the ability to aggregate 1000's of OSDs, one of the critical questions we sought to answer was "how widely should we stripe files?" Ideally, we would prefer to stripe a file as widely as possible. Wider stripes increase the number of OSDs involved in a transfer and hence increase the bandwidth available to large files or clusters of smaller files accessed in parallel.

When we ran tests that varied the stripe width we discovered, as expected, that bandwidth increased with increasing number of OSDs. Figure 2 shows this result with a single client reading from a single file that is striped over N OSDs ($3 <= N <= 31$). The data demonstrates linear scaling as the number of OSDs increases from 3 to 7, with 7 OSDs achieving an aggregate bandwidth of 87 MB/sec for the single client (we used a single client to avoid any of the non-linear effects caused by concurrent streams, as discussed in Section 5).

However, we also discovered that if we continued to increase the number of OSDs past 14, ***aggregate bandwidth actually decreases*** (Figure 2). This may seem counter-intuitive because striping across more OSDs increases the potential bandwidth of the system. However, wider striping increases the potential amount of traffic simultaneously *incast* into the network. With each receiver limited to a single Gigabit Ethernet link, and the Ethernet switches bounded buffer capacity, too many concurrent senders can overwhelm the network. Specifically, traffic between the OSDs and client will begin to backup into the network. For switches with sufficient buffer space, small bursts can be buffered in the network. But for large concurrent senders, the network switch is forced to drop packets[1], resulting in TCP timeouts and retransmitted data.

Interestingly, this behavior is not seen when running a typical streaming network benchmark, such as netperf. Running netperf on the same client and OSD hardware used to collect the data for Figure 2, we saw client receive bandwidth quickly grow to over 100 MB/sec and then level off w***ith no decrease in performance as we increased the number of senders***[2] ***from 3 to 40***. This stability in peak bandwidth is because there is no synchronization in the streaming workload; when some streams incur packet loss and stall waiting for TCP's retransmission algorithm to kick in, other streams may continue to use all of the available bandwidth. Storage's behavior is fundamentally different; requests across OSDs are synchronized by the client (i.e., each OSD stream constitutes one or multiple stripe units within the `read()` system call). Hence, unlike streaming workloads such as netperf, where different streams can utilize the bandwidth at different times, the file system/application must wait until all OSDs have returned their chunk of data before proceeding. Therefore, if one storage stream pauses (e.g., due to packet loss), the entire client application must wait.

Figure 3 illustrates the incast behavior on the network. The figure plots the TCP sequence numbers and retransmissions when a single client is reading from a file striped across 30 OSDs. During the run, each OSD experienced 2K to 3K duplicate ACKs and 200 to 400 retransmit timeouts. With TCP's fast retransmit algorithm, duplicate ACKs quickly recover single packet loss. Severe incast, however, causes multiple packet drops (often consecutive packets), forcing the system to wait for retransmit timeouts. During this time the client must sit idle, waiting for the one or two straggler OSDs to retransmit their data. For example, if a client requests 10 MB of data from the OSDs, the total time required to read the data is: A) 10 msec to read the data from the disk platters (333K per OSD) plus B) 10 msec for the client to receive 10 MB, for a total of 20 msec. If just one OSD experiences enough packet loss to result in one 200 msec TCP retransmit timeout, the system suffers a 10X decrease in performance.

1. Another solution is for Ethernet switches to generate link-level flow control. However, if multiple senders sit behind a GE link (or set of trunked links), link-level flow control can actually hurt performance by penalizing senders that are not contributing to buffer overflow.
2. We stopped running the experiment at 40 senders.

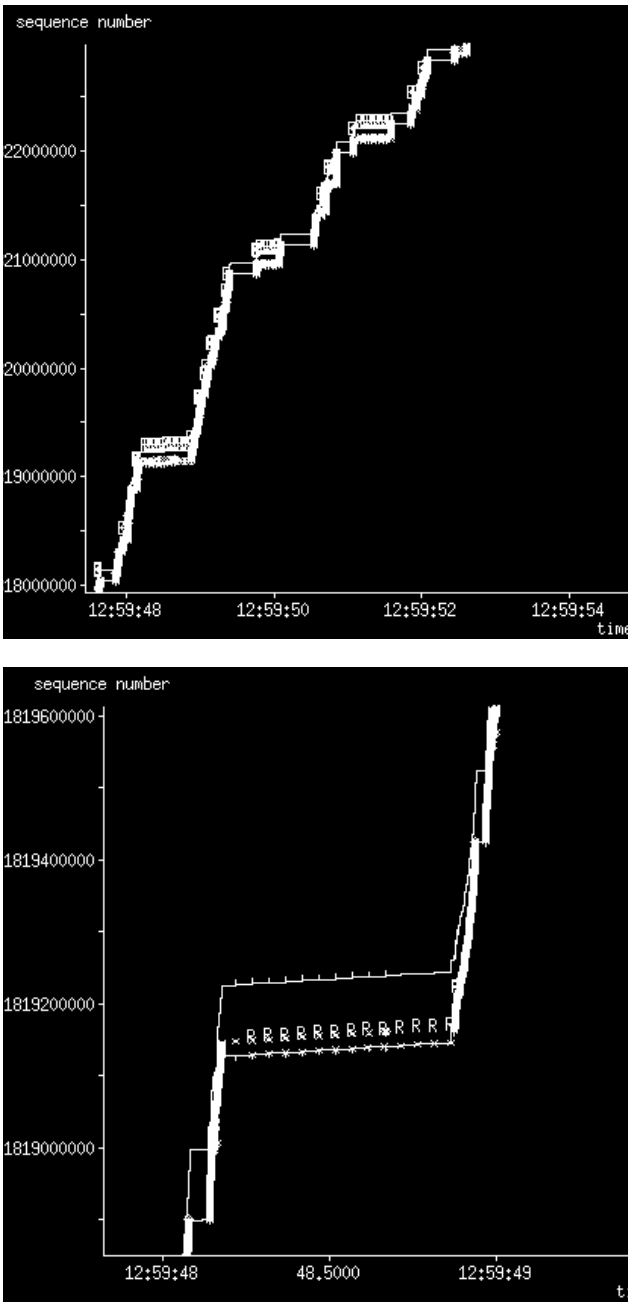

Figure 3: TCP Sequence Plot from OSD. Reading a file striped across 30 OSDs can overwhelm the network buffers in front of the reading client to cause significant packet loss. The client's receive socket buffer was 256KB and the total read throughput was 20 MB/sec. In contrast, a client with only 64KB receive socket buffers reading from 10 OSDs can receive data at over 80 MB/sec. The data was captured by running tcpdump on each of the OSDs over a 60 second dd test where the client was reading data from all of the OSDs. The figure was generated using tcptrace. The red "R"s represent retransmitted packets. The bottom plot is an enlargement of the top plots first second.

The basic problem is not the packet loss itself, but that TCP flow-control algorithms were originally designed for WANs and LANs. Panasas uses a low-latency, high-bandwidth GE network. This means that TCP timeouts on the order of 10's to 100's of msecs can cause dramatic loss in performance for storage. Streaming workloads minimize this problem because they are always trying to inject data into the network, allowing them to quickly learn through TCP's feedback algorithms what a "fair" transmission rate is. Network-attached storage is far burstier and has a more difficult time learning what a "fair" transmission rate should be.

TCP, however, is very robust and even with its msec-scale time-outs and lack of synchronization across multiple senders, it is possible to tune TCP to avoid the incast problem and achieve high bandwidth within a storage network. This is very important because storage is rapidly adopting TCP/IP and Ethernet as a transport [iSCSI]. To mitigate the high packet loss we tried several solutions. First, we reduced the retransmit timeout from 200 msec down to 50 msec. TCP remained stable with this shorter timeout and greatly decreased the performance costs associated with multiple packet loss. Second, we implemented TCP's Selective Acknowledgement (SACK) option inside our OSDs to enable quick recovery from packet loss [IETF96]. This resulted in a modest performance gain. However, Ethernet switches' commonly-used tail drop policy [Peterson00], where switch buffer limits force multiple sequential packets from the end of a stream (flow) to be dropped, often results in packet loss at the end of a stream, something SACK is unable to recover from.

Our most effective tuning parameter was the receive socket buffer size. Before discovering the incast problem, we believed that high bandwidth required a large window size. Aggregating OSDs and the accompanying incast effect taught us that while a client must have a large aggregate receive buffer size, each individual stream's receive buffer should be relatively small. Therefore, we reduced the clients (per OSD) receive socket buffer size to under 64K [Semke98]. This effectively reduced each OSDs transmit window, decreasing the likelihood that concurrently transmitting OSDs would overwhelm the switch buffering capacity.

To provide the highest bandwidth possible PanFS stripes data across all available OSDs. However, to avoid massive retransmission caused by an incast traffic pattern, PanFS implements a two level striping pattern as illustrated in Figure 4. The first level of striping, based on the stripe units previously discussed, is optimized for RAID's parity update performance and overhead. The second level of striping is designed to resist incast induced bandwidth penalties by stacking successive parity stripes in the same subset of objects. We call the N sequential parity stripes that are stacked in the same set of objects a *visit* because a client repeatedly fetches data from just a few OSDs (whose number is controlled by the parity stripe width) for a while, then moves on to the next set of OSDs. This striping pattern resists incast problems because in most cases the number of OSDs a client is communicating with at any given time is limited to the width of the

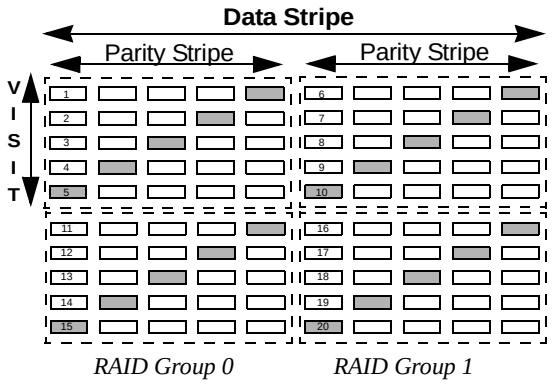

Figure 4: Two Level Striping Layout**.** PanFS stripes data across all of available OSDs, but limits most simultaneous accesses to the number of OSDs in a parity stripe to avoid incast. In the figure, each small block represents a stripe unit and each column depicts all of the data stored on an OSD (parity blocks are shown in gray, and the stripe layout order is numbered 1 through 20). In this example, each parity stripe is 5 OSDs wide and the visit size is 5 parity stripes. The two parity stripes of this file's layout utilize all 10 OSDs.

parity stripe. Typically, we stripe about 1GB of data per visit, using a round-robin layout algorithm of visits across all OSDs.

5. Aggregation of OSDs and RAID

To leverage the aggregate performance of all the OSDs in the storage cluster, PanFS stripes data across all the OSDs. The default striping algorithm for large files is RAID 5 (64K stripe units), with each file striped across as many OSDs as possible, using the per-file RAID groups described in Section 4. The MDS selects the OSDs over which to stripe a file, partitioning the OSDs into RAID parity groups of up to 13 (max). The MDS also closely monitors space and load, creating file-to-OSD mapping assignments that evenly distribute the load across all OSDs in the cluster. The PanFS client module implements RAID 0, 1 and 5 (RAID 5 is the default for large files), making the client responsible for computing and updating parity on writes. Because writes must send both the data and parity to the OSDs, writes will typically incur about an 11% bandwidth overhead[1] for RAID 5 files.

Figure 5 shows how the bandwidth scales linearly with the number of clients; as we scale from 10 to 60 clients, the graph peaks at 3,100 MB/sec for reads and 2,700 MB/sec for writes. Although the testbed used to collect the data presented here was limited to 10 shelves and 60 clients, we have run a similar test on 30 shelves and 150 clients and observed similar linear scaling up to 10 GB/s. For these tests, each shelf was connected to a Dell 5224 GE switch 4-port trunked to an Extreme Blackdiamond

1. All write bandwidth numbers compute bandwidth based on (total number of data bytes / total time) instead of (data+parity/total time). Therefore, write numbers have an 11% overhead.

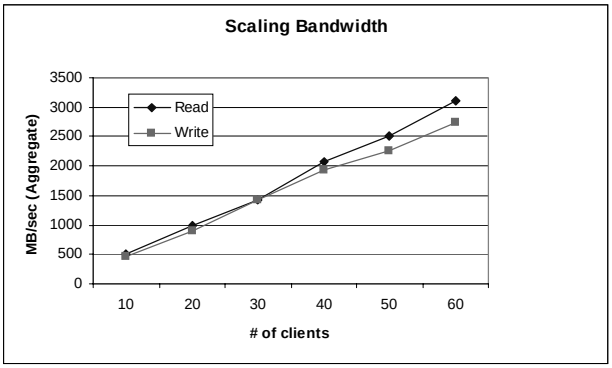

Figure 5: Scalability Across the Storage Cluster. A 100 OSD storage cluster's read and write bandwidth when {10, 20, ..., 60} clients concurrently access unique files within the cluster.

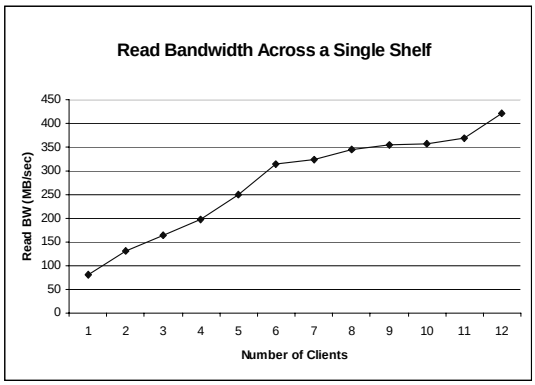

Figure 6: Scalability Across a Small Storage Cluster. A 10 OSD storage cluster's read bandwidth when 1 to 12 clients concurrently access unique files with the cluster.

6816 switch. Also, each Dell 5224 switch aggregated up to six 2.4 GHz P4 Linux workstations into the network fabric.

Figure 6 shows a similar picture, using 1-12 clients across a single 10-OSD shelf. Performance scales linearly for 1 to 6 clients, continuing to increase at a slightly lower rate from 7 to 12 clients. This sub-linear scaling is largely due to OSD transfer scheduling. As the number of concurrent clients accessing a group of OSD increases, the order in which each OSD services a client's request will differ. OSD request ordering is determined by a combination of command arrival order and whether the request hits or misses in the read-ahead cache. Different client requests may become intermingled in the network or within the OSD, resulting in different response times from the various OSDs. Moreover, as the number of streams increases the read-ahead cache's hit rate decreases slightly. Because the OSD will always try to service requests that hit in the read-ahead cache, the probability of command reordering grows as the hit rate drops. Therefore, as the number of clients increases, the system continues to provide increased aggregate throughput, but clients individually see a reduction in their own bandwidth.

6. Scalable File Systems

There are numerous recent examples of other file systems designed to deliver varying degrees of scalable performance. One of the key differentiators is between in-band and out-of-band solutions. The Global File System (GFS), a fully symmetric distributed file system, is one example of an in-band solution [Preslan99]. In GFS, each node of the cluster shares full access to disk storage, requiring that the nodes negotiate temporary ownership of any storage block to ensure consistency. To make changes in file data, a client obtains ownership and up-to-date copies of file data, metadata and associated directory entries. To allocate space or move data between physical media locations, a client obtains ownership of encompassing physical media and up-to-date copies of that media's metadata. Unfortunately allocation is not a rare event, so even with minimal data sharing, inter-node negotiation over unallocated media is a common arbitration event. Because of the central nature of arbitrating for ownership of resources, GFS and systems like it have distributed lock managers that employ sophisticated techniques to minimize the arbitration bottleneck.

One limitation of GFS and other in-band solutions is their limited ability to enable direct access from non-GFS cluster nodes. Out-of-band file systems, such as IBM's Sanergy and EMC's High Road [EMC, IBM], overcome this limitation by differentiating the capabilities of file system code running on cluster nodes from the file system code running on I/O nodes: only I/O node file system software can arbitrate the allocation and ownership decisions and only this software can change most metadata values. The file system software running on cluster nodes is still allowed to read and write storage directly, provided it synchronizes with I/O nodes to obtain permission, up-to-date metadata and all allocation decisions and metadata changes. Because metadata control is not available in the data path from the cluster node to the storage device, these I/O nodes are called metadata servers or out-of-band metadata servers. Metadata servers can become a bottleneck because their block abstraction is so simple that many cluster write commands will require synchronization with metadata servers [Gibson98].

Further, the isolation of metadata control on the I/O nodes is by convention only; the storage interface will allow any node that can access it for any reason to execute any command including metadata changes. There is no protection from accidental or deliberate inappropriate access. Data integrity is greatly weakened by this lack of storage enforced protection; the block interface doesn't provide the fundamental support needed for multi-user access control that is provided.

7. Conclusions

There is a wide a growing number of high-performance computing applications that can directly benefit from the scalable bandwidth PanFS delivers, including:

- Energy research, simulation and high-energy physics
- Earth, Ocean and atmospheric sciences, global change and weather prediction
- Seismic data analysis
- Large scale signal and image processing
- Automotive design and simulation
- Digital media applications

Each of these applications process massive amounts of data and increasingly rely on cluster systems to provide the computation, networking and storage resources necessary to scale. PanFS supports these applications by delivering both scalable per-client bandwidth and scalable storage across the cluster.

Key to achieving scalable bandwidth of PanFS has been the careful design of the Object-based Storage Device (OSD). Our results show how the Panasas OSD leverages high-level information exposed by the OSD interface, allowing the OSD to carefully manage data prefetching and the OSD cache. Further, by tracking information based on object-ids, Panasas is able to tune our algorithms to accommodate a large number of clients. This is critical for applications such as Blast, where each client independently reads from the same set of files. The OSD is able to track each independent client's request stream. Further, by striping data cross all of the OSDs, but chunking the data into parity groups, PanFS's layout is able to distribute the load of 1000's of clients across the storage system.

Our results also show that network-attached storage requires a careful coupling with the network. Specifically, PanFS traffic is much burstier than streaming workloads and the typical high-bandwidth network benchmarks commonly available [iperf, ttcp]. The incast behavior described in Section 4 underscores that too much scalability in the "wrong dimension" can overwhelm the network. Instead, by carefully limiting our striping width and tuning TCP's socket buffer sizes, we are able to maintain peak overall aggregate performance.

8. References

[Azagury02] Azagury, A., Dreizin, V., Factor, M., Henis, E., Naor, D., Rinetzky, N., Satran, J., Tavory, A., Yerushalmi, *Towards an Object Store*, IBM Storage Systems Technology Workshop, November 2002.

[Boden95] Boden, N.J. et al, "Myrinet: A Gigabit-per-Second Local Area Network", IEEE Micro, Feb. 1995.

[Cisco] http://www.cisco.com

[Clark89] Clark, D.D. et al., "An Analysis of TCP Processing Overhead," IEEE Communications 27,6 (June 89), pp. 23-36.

[EMC] http://www.emc.com/techlib/

[Extreme] http://www.extreme.com

[Gibson92] Gibson, G., “Redundant Disk Arrays: Reliable, Parallel Secondary Storage,” MIT Press, 1992.

[Gibson98] Gibson, G., et al. A cost-effective, high-bandwidth storage architecture.In *Proceedings of the ACM 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (San Jose, Calif., Oct). ACM Press, New York, 1998, 92–103; see also www.pdl.cmu.edu.

[Gobioff96] Gobioff, H. et al., “Security for Network-Attached Storage Devices,” CMU-CS-96-179, 1996.

[Hartman93] Hartman, J.H. and Ousterhout, J.K., “The Zebra Striped Network File System”, 14th SOSP, Dec. 1993.

[IBM] www.ibm.com/software/tivoli/products/sanergy/sanergy-tech.html

[IETF] http://www.ietf.org/html.charters/ips-charter.html

[IETF96] Mathis, M., et. al., TCP Selective Acknowledgement Options, RFC 2018, www.ietf.org.

[iperf] http://dast.nlanr.net/Projects/Iperf

[Lustre03] *Lustre: A Scalable, High Performance File System*, Cluster File System, Inc. 2003. http://www.lustre.org/docs.html

[Myricom] http://www.myricom.com

[OSD] Draft OSD Standard, T10 Committee, Storage Networking Industry Association (SNIA), ftp://ftp.t10.org/t10/drafts/osd/osd-r09.pdf

[Panasas] http://www.panasas.com

[Pease02] Pease, D, Rees, R., Hineman, W, et al., IBM Storage Tank™: A Distributed Storage System, in Proceeedings of the 1st USENIX Conference on File and Storage Technologies Work In Progress (FAST-WIP), Monterey CA, January 2002.

[Peterson00] Larry L.Peterson and Bruce S. Davie, “Computer Networks, A Systems Approach,” Morgan Kaufmann Publishers, 2000, San Francisco, CA.

[Preslan99] Preslan, K. W., O’Keefe, M.T., et. al.., A 64-bit shared file system for Linux, in Proceeedings of the 16th IEEE Mass Storage Systems Symposium, 1999.

[Semke98] Semke, J, Mahdavi, J. and Mathis, M. "Automatic TCP Buffer Tuning", *Computer Communications Review*, a publication of ACM SIGCOMM, volume 28, number 4, October 1998.

[Sistina] http://www.sistina.com/products_gfs.htm

[ttcp] http://www.pcausa.com/Utilities/pcattcp.htm

[Traw95] Traw, C.B.S. and Smith, J.M., “Striping Within the Network Subsystem”, IEEE Network, Jul./Aug. 1995.

[vanMeter98] Van Meter, R., Finn, G., and Hotz, S. VISA: Netstation’s virtual Internet SCSI adapter. In *Proceedings of the ACM 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (San Jose, Calif., Oct.). ACM Press, New York, 1998, 71–80; see also www.isi.edu/netstation/.

[Voltaire] http://www.voltaire.com

[Watson95] Watson, R.W., and Coyne, R.A., “The Parallel I/O Architecture of the High-Performance Storage System (HPSS),” 14th IEEE Symposium on Mass Storage Systems, Sept. 1995, pp. 27-44.

[Wilkes95] Wilkes, J. et al., “The HP AutoRAID Hierarchical Storage System”, 15th SOSP, Dec. 1995.

[Wiltzius95] Wiltzius, D. et al., “Network-attached peripherals for HPSS/SIOF”, http://www.llnl.gov/liv_comp/siof/siof_nap.