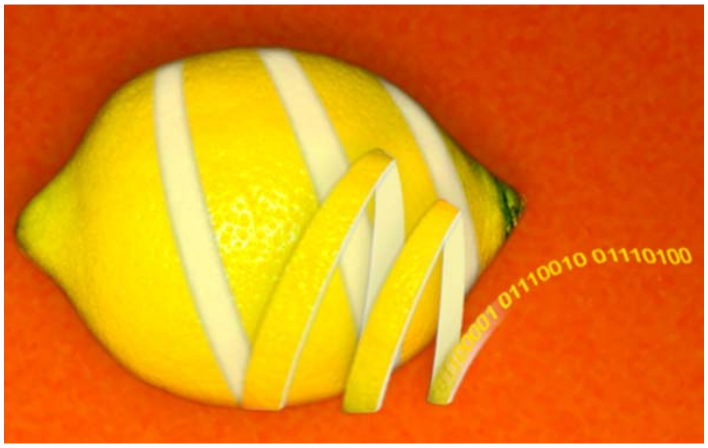
1 Title

Zest: The Maximum Reliable TBytes/sec/$ for Petascale Systems



2 Team Information

The Pittsburgh Supercomputing Center
Advanced Systems Group:
Nathan Stone, Doug Balog, Paul Nowoczynski,
Jason Sommerfield, Jared Yanovich

3 Abstract

PSC has developed a prototype distributed file system infrastructure that vastly accelerates aggregated write bandwidth on large compute platforms. Write bandwidth, more than read bandwidth, is the dominant bottleneck in HPC I/O scenarios due to writing checkpoint data, visualization data and post-processing (multi-stage) data. We have prototyped a scalable solution on the Cray XT3 compute platform that will be directly applicable to future petascale compute platforms having of order 10^6 cores. Our design emphasizes high-efficiency scalability, low-cost commodity components, lightweight software layers, end-to-end parallelism, client-side caching and software parity, and a unique model of load-balancing outgoing I/O onto high-speed intermediate storage followed by asynchronous reconstruction to a 3rd-party parallel file system. The absence of a central metadata service further reduces latency, allowing for the maximum reliable performance per unit cost for petascale systems.

4 Problem Statement and Hypothesis

Computational power in modern High Performance Computing (HPC) platforms is rapidly increasing. Moore's Law alone accounts for doubling processing power roughly every 18 months. But a historical analysis of the fastest computing platforms by Top500.org shows a doubling of compute power in HPC systems roughly every 14 months[1], so that the first petaflop computing platform is expected in late 2008. This accelerated growth trend is due largely to an increase in the number of processing cores; the current fastest computer has roughly 65K cores. An increase in the number of cores imposes two types of burdens on the storage subsystem: larger data volume and more requests. The data volume increases because the physical memory per core is generally kept balanced resulting in a larger aggregate data volume, on the order of petabytes for petascale systems. But more cores also mean more file system clients, more I/O requests on the storage servers and ultimately more seeking of the back-end storage media while storing that data. This will result in higher observed latencies and lower overall I/O performance.

Many HPC sites implement parallel file systems comprised of an increasing number of distributed storage nodes. As the number of clients continues to increase and to outpace the performance improvement trends of storage devices this solution will necessitate additional measures.

Our system achieves high performance at low cost by striving for the maximum disk drive efficiency scaling across all drives in the storage system. This is accomplished with three features.

First, Zest recovery information, parity, is generated at the client. Since parity (RAID) calculations are computationally expensive and memory bandwidth intense, having all cores of a running application available to perform these calculations removes this overhead from the server. Once the parity is calculated and included in the data sent from the client, Zest makes sure that when written to disk, the writes are to "RAID Vectors" of drives. "RAID Vectors" are groups of drives that maintain the

RAID integrity.

Second, when writing to disk, sequential I/O is required to maximize performance. In traditional file systems, the block allocators lose performance when trying to handle many client requests at once. The Zest block allocator ushers the incoming data onto the next available disk drive. Since the data from a single client will be fragmented across many disks, the reassembly information is carried with the data. To further enhance performance, Zest preferentially uses the outer cylinders of each disk drive.

Finally, in order to give the user a file system view of their data, the file fragments are reconstructed and written as regular files into a third party parallel file system, such as Lustre.

In this way we both reduce the running application's perception of contention and latency and substantially increase the achieved bandwidth.

Based on our previous experiences with custom storage infrastructures[2] and new prototype development, we are creating a scalable distributed solution that will scale to of order millions of clients and impose only of order 10% overhead relative to the per-disk peak write speed using commodity storage hardware components.

5 Data and Storage Pipeline

There are four locations where application data resides on its path through the Zest architecture, associated with four specific states. The states are "cached", "incoming", "full parity", and "completed". These states are depicted in Figure 1.

The first location is the client-side memory cache, managed in blocks, which have a configurable size. In "small" write scenarios (data size less than the block size) application data writes are intercepted by the client library, which copies data into the memory cache. This is the "cached" state. During this time the client library also calculates a parity block for each group of data blocks, allowing for a high level of software RAID and recovery. The client library sends data by RDMA to memory on the Zest servers when a) smaller coalesced writes have filled a cache block; b) incoming writes are larger than the cache block size (i.e. "zero-copy") or; c) caching is deactivated. In the case of a system such as the Cray XT3, this network traffic is routed through the SIO nodes to the Zest servers.

When data or parity blocks arrive in memory on a Zest server, which runs on Zest IO Nodes or *Zestions*, the blocks are in the "incoming" state. They wait in memory for transfer to disk by one of the Zest server's many I/O threads. There is one thread for each disk. These threads copy data from "incoming" memory to custom-formatted partitions on their associated disk whenever the disk is able to take the data. The "full parity" state is achieved when all data and parity blocks in a given set are stored in these custom partitions.

In this manner, application data from compute clients are distributed among the Scalable Units on a first-come, first-served basis. For the data to be accessible by the user, all data fragments residing on SU disks must be reassembled and written to a file system. A small amount (of order 0.02%) of metadata is stored with the bulk data for use in this reassembly phase. During reassembly, a low priority thread within the Zest server asynchronously issues requests to read data back from the custom partitions and reassembles it into a parallel file system with correct offsets into the output files originally named by the user. When this process is completed, the application data are in the "completed" state.

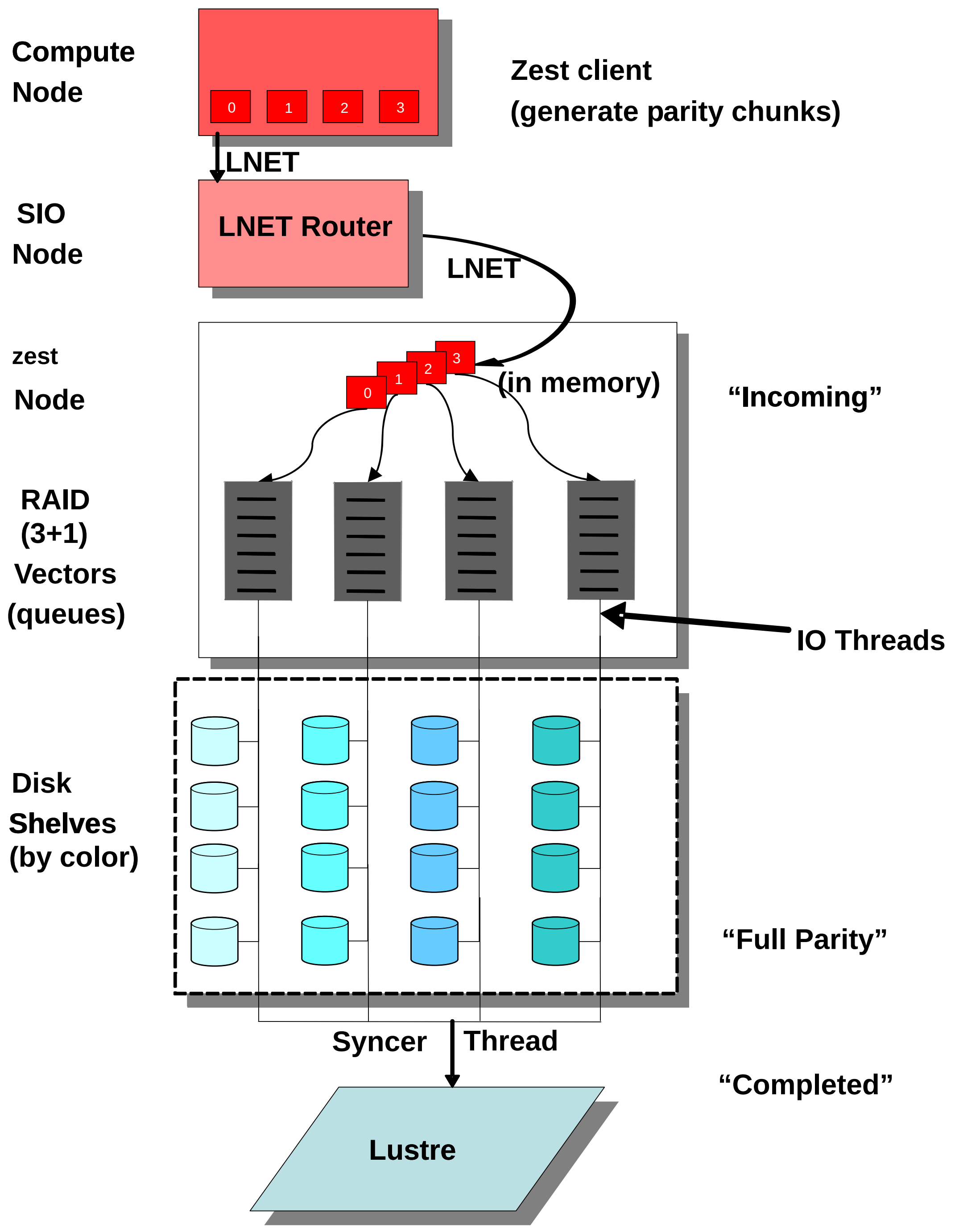


Figure 1: Zest architectural design showing hardware components (labeled at left), software components (center details) and the state model (labeled at right).

6 Description of the Solution to the Problem

6.1 Zest Server Design Concepts

In the age of Petascale compute systems, where disk speeds are greatly outpaced by the performance of CPU's and speed of high-performance networks, maintaining sequentiality at the spindle is the most effective method of providing a scalable, cost-effective checkpoint IO system. Zest is designed to perform sequential I/O whenever possible. To achieve a high degree of sequentiality, Zest's block allocation scheme is not determined by data offset or the file object identifier but rather the next available block on the disk. This simple, non-deterministic data placement method is extremely effective for providing sequential data streams to the spindle and minimizing costly seeks. The sequentiality of the allocation scheme is not affected by the number of clients, the degree of randomization within the incoming data streams, or the RAID attributes (i.e. parity position) of the block. It should be noted that a block's parity position does restrict the number of disks which may handle it. This is the only determinism maintained in the write process.

Prior to the advent of petascale computing this data storage method would be considered prohibitive because it destroys the convenience of inference at two critical layers distributed file system metadata and the block level RAID, thus increasing management overhead at those points. Object-based parallel file systems (Lustre) use file-object maps to describe the location of a file's data. These maps are key components to the efficiency of the object-storage method because they allow for arbitrary amounts of data to be indexed by a very small data structure composed merely of an ordered list of storage servers and a stride. In essence, the map describes the location of the file's sub-files and the number of bytes which may be accessed before proceeding to the subfile or stripe. There are several caveats of this method, the most obvious being that the sub-files are fixed and are managed by an underlying block-managed file system such as Ext3. GPFS uses a different method for indexing data at the IO node but has the same limitation of fixed sub-file placement. Sub-file management increases complexity at the spindle because the block allocation schemes cannot guarantee sequentiality in the face of thousands or millions of simultaneous IO streams.

Hardware RAID systems infer that every same numbered block of its constituency of spindles are bound together to form a protected unit. This method is effective because only the address of a failed block is needed to determine the location its protection unit cohorts. Despite this advantage, we have found that hardware RAID systems located at the fileserver do not deliver a high degree of spindle efficiency and thus more spindles are needed. Within the context of a Petascale system the number of spindles needed for hardware RAID becomes prohibitive largely because of cost but also because of rebuild times.

6.2 Zest Client Workings and RAID Pre-processing

The client library transparently intercepts I/O-related system functions on compute nodes, directing those with designated paths (e.g. "/zest/my/output/file") toward the Zest SUs while allowing other I/O (e.g. "/home/my/path/") to pass straight through to the ordinary system I/O functions with negligible overhead. The Zest client library has two primary tasks:

Write Aggregation – Merge smaller I/Os into larger data blocks cached in client node memory. This reduces and effectively amortizes the communications overhead associated with tiny I/O operations and also minimizes lock-related contention in the Zest server.

Parity and CRC Calculation – Calculate parity at the client side to protect against some server-side failure modes. Performing this calculation at the client side also distributes this workload across a larger number of cores, optimizing the compute resource performance as a whole and allowing the Zest server to focus on data throughput.

6.3 Zest Client to Server Data Transfer

Zest servers are equally accessible by all compute processors and support dynamic fail-over through the redundant SU architecture. Zest servers are load-balanced based on the modulus of the compute processor identifier. A single compute processor may access multiple Zest servers sequentially, but its data output is not striped across the Zest servers.

The Zest client-side network library has the ability to send buffers synchronously or asynchronously depending on the IO pattern or the buffer type. For example, the client's parity accumulator buffer is sent asynchronously to the server while a multi-MB write buffer will be mapped into several parallel, zero-copy, blocking requests.

Adapting the Cray XT3 I/O environment, the Zest server cluster is located on an InfiniBand cloud external to the machine. The SIO nodes act as a gateway between the XT3's internal network and the Zest servers. We run a Lustre LNET routing service on each of the SIO nodes to route traffic from the internal high-speed compute interconnect to the external InfiniBand network, providing seamless connectivity from compute processors to the external Zest storage and services.

To achieve compatibility with the Lustre routing service, Zest's networking library is largely based on Lustre's LNET and RPC subsystems. The Zest server uses a modified TCP-based Lustre networking driver with additional support for InfiniBand sockets-direct protocol. Since Zest is a user mode service, it does not have access to the native, kernel-mode InfiniBand Lustre driver so SDP was chosen for its convenient path into the InfiniBand interface.

6.4 Zest Server Fault Handling and Reconstruction

Media Error Handling

All Zest data and metadata are protected by a 64 bit CRC used to detect media errors. All CRCs, except for the on-disk metadata are calculated at the client, data CRCs are not verified by the server for performance reasons. Only on read will the Zest server perform CRC operations to ensure that the block has not been compromised. When a bad block is found its parity group information is located via a lookup into a separate device. This device, aptly named “parity device”, is a solid state memory device whose purpose is to maintain the parity group structure for every block on the system. Any Zest block's parity group descriptor is located via a unique address composed of the block's disk and block identifiers. Since the IO pattern to the parity device is essentially random it has been outfitted with a small solid-state disk. Currently an 8 million block Zest system requires a 4 gigabyte parity device. The parity device update path is asynchronous and therefore, if needed, the entire device may be reconstructed during file system check.

Run-time Disk Failures

Since the Zest system manages RAID and file-objects, handling of disk failures only requires that volatile blocks are rebuilt. Zest has full knowledge of the disk's contents so old or unused blocks are not considered for rebuilding. During the course of normal operation, Zest maintains lists and indexes of all blocks being used within the system. In the event of a disk failure, the set of blocks who have not been synchronized or whose parity group cohorts have not been synchronized will be rebuilt. Here the parity device will be used to determine the failed block's parity group cohorts. It must be noted that Zest cannot recover from simultaneous failure of a parity device and a data disk.

File System Check and Boot-time Failures

On boot, the file system check analyzes the system's parity groups and schedules the synchronization of volatile blocks. If a disk fails in the start-up phase any volatile blocks which it may have been holding are located during the fsck process and rebuilt.

7 Software Used

The Zest infrastructure is entirely new software, built by PSC's Advanced Systems Group for large compute platforms. The only non-original component in the Zest code base is a 3rd-party communication library. We chose the Lustre LNET RPC library as our communication library because of its portable support for RDMA over multiple network types and also its ability to route data between heterogeneous networks. And we chose Lustre as the parallel file system because of its highly scalable design and ongoing support.

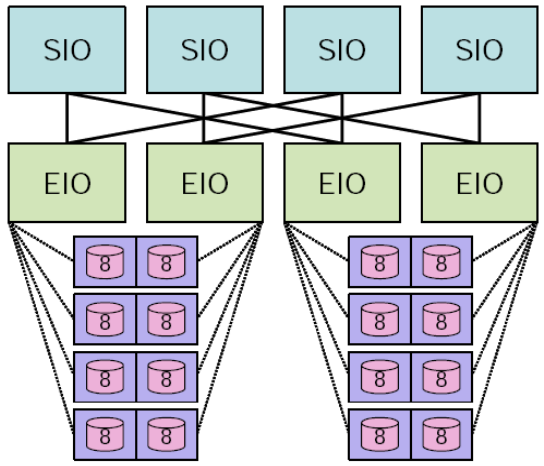
We ran performance and scalability benchmarks on Zest and compared those to Lustre, a popular high-performance, commercial parallel file system. Because of Zest's transparent intercept library we were able to build all benchmarks from unmodified sources, as distributed. The tests applications included IOR[3] (using the POSIX API), which arose out of the Scalable I/O Project[4] at Lawrence Livermore National Laboratory and was subsequently incorporated into the ASCI Purple benchmarks. We have also run with SPIOBENCH[5], which arose out of the NSF Petascale computing initiative (NSF project 05-625)[6]; Bonnie++[7], a well-known UNIX file system benchmark that does not require MPI; and FIO, our own MPI-based parallel I/O regression test suite.

8 Hardware Configuration

Application clients ran on both Linux and Cray XT3 compute environments. The XT3 compute nodes run a micro-kernel OS, anticipating the structure of future petascale machines. Our Linux environment supported only up to 128 clients, but the routing supported the SDP protocol over multiple server interfaces. Our XT3 environment allowed scalability testing up to thousands of nodes, and the routing supported native (o2ib) protocol for Lustre traffic and TCP-over-IB protocol for Zest to a single server interface. This is a limitation we expect to lift in time.

For the XT3, the client library runs on compute nodes, sending messages to the storage servers via the compute resource’s Service and I/O (SIO) nodes. The SIO nodes route Zest traffic between the internal network and their connections to an external, InfiniBand network to which the Zest storage servers are also directly connected. These Zestions have multiple InfiniBand interfaces and share multiple disk enclosures with a failover partner. A pair of Zestions and their associated disks are collectively referred to as a *Scalable Unit* (SU), see Figures 2a and 2b. Commodity SATA disks are used to optimize cost-effectiveness, though their enclosures are directly attached to each Zestion of the SU via Serial Attached SCSI (SAS) links. As such each Zestion is capable to accessing all of the SU’s drives in the event of its partner failure, avoiding any single point of failure. Each Zestion, though part of an SU, services requests independently without the synchronization of a central metadata service or other components in the SU absent the failure of its peer. This compute system connectivity model of utilizing gateway nodes connected between internal compute nodes and external service nodes is expected to persist in petascale compute resources.

The SIO node hardware itself is dictated by the compute platform vendor. For our reference implementation these are dual-core Opteron processors running Suse Linux within the UNICOS/lc framework. They are connected to the Zestion nodes via 4X single data rate InfiniBand links to the PSC InfiniBand storage fabric.

a)


b)

Figure 2 a)"Scalable Unit" storage components and connectivity. b) Scalable Unit hardware (left) connected to PSC’s Cray XT3 (right).

Current Zestions are dual socket, multi-core SMPs with dedicated 4X SAS links to each of four disk enclosures providing enough bandwidth to stream to all of the drives simultaneously. Each Zestion is configured to normally use half of the available drives on each shelf, leaving the other drives for its peer. Our current prototype has one 16 GB solid state disk four shelves of 750GB capacity drives each rated at 75 MByte/sec write bandwidth. This yields a peak theoretical streaming rate of 1.2 GB/s and raw disk capacity of nearly 12TBytes per Zestion. Assuming a modest technology progression over the next few years, a petaflop HPC system would require of order hundreds of SUs to obtain 1 TByte/sec aggregate write bandwidth.

No hardware RAID controllers were involved, as Zest does not require them. All hosts were InfiniBand connected, although the tests used IB-native, SDP and TCP-over-IB protocols, as specified below.

Zest assigns each of the hard drives to an I/O thread and uses the solid state drive to store parity set related metadata. This "parity drive" does not hold parity data per se, but only what is necessary to re-associate blocks into parity groups.

For comparison, we ran applications writing to Lustre in two configurations: "RAID0" and "RAID5". In the RAID0 case Lustre presented the 16 hard drives as 16 Object Storage Targets (OSTs) and each client "striped" data across all OSTs without parity generation. In the RAID5 case Lustre presented a single OST composed of a single Linux software RAID volume. Both configurations used freshly formatted disk partitions identical to those used by Zest.

9 Experiments, Measurements and Quantitative Results

Here we present aggregate write bandwidths as measured and reported by IOR. Unless otherwise denoted as "backend", bandwidths are reported as measured by the application. Backend volume and bandwidth are naturally higher, accounting for additional parity information.

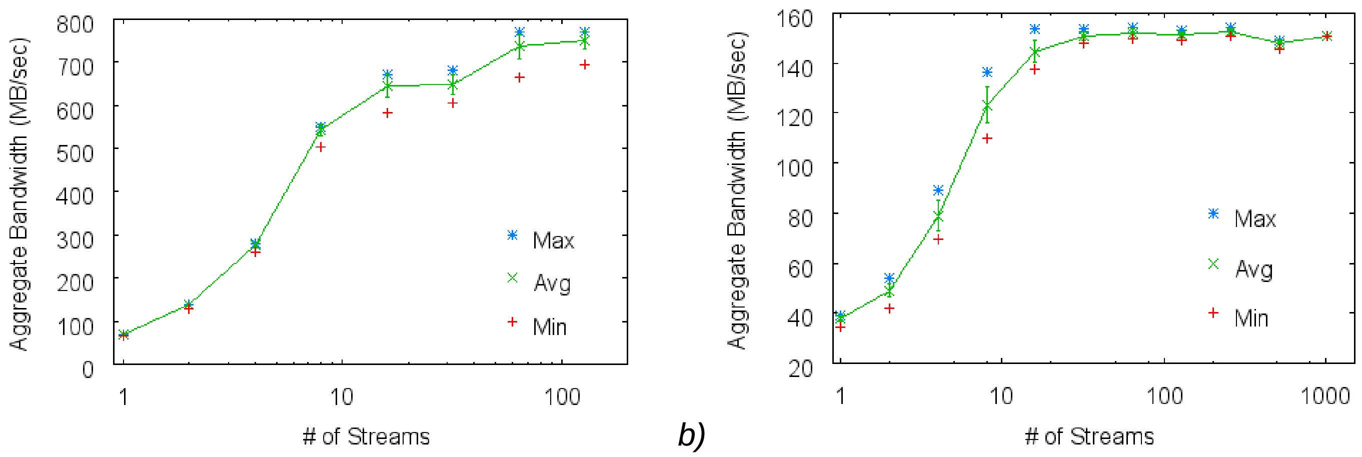


Figure 3: Zest aggregate application write bandwidth scaling on a) Linux and b) XT3 compute platforms.

Figure 3a shows Zest scaling of aggregate application bandwidth versus the number of client streams under Linux up to 128 clients. We plot the average values with error bars as well as minimum and maximum values to show the consistency of performance. The aggregate raw spindle speed of all disks in the server is 1200 MB/sec. Zest achieves a significant fraction of that rate even for as few as 8 streams. At higher stream counts the application observes an average rate in excess of 750 MB/sec -- over 63% of the server's raw spindle bandwidth. Backend measurements during these tests averaged over 800 MB/sec, consistent with the 15+1 client RAID configuration used in testing. Peak backend bandwidths have reached as high as 1.1 GB/sec -- over 91% of the server's raw spindle bandwidth.

The maximum CPU consumption observed throughout these tests was 50%. This corresponded to usage of the SDP protocol, which requires implicit buffer copies. During non-SDP traffic the server CPU was generally more than 80% idle. Furthermore the server's memory buffers, used to stage data from the network to the disks, remained virtually empty at all times, confirming the fact that these performance rates are fundamentally network-limited.

Figure 3b shows Zest scaling under XT3 up to 1024 clients. Due to the limited TCP-over-IB routing these rates are severely network limited. However scaling out to 1024 clients is smooth and consistent, showing no problems for the RPC handling infrastructure.

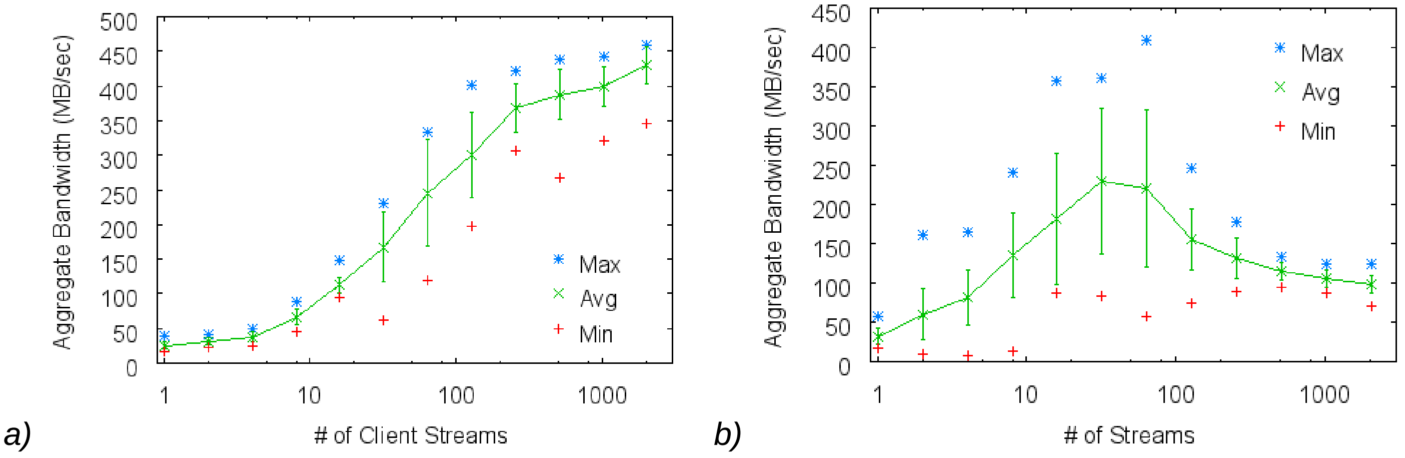
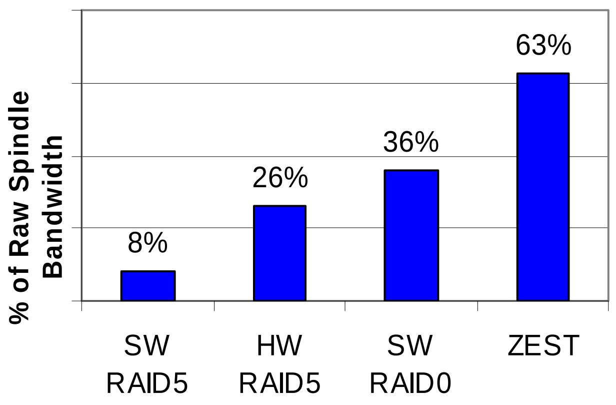


Figure 4: Lustre aggregate application write bandwidth scaling for a) RAID0 and b) RAID1 configurations.

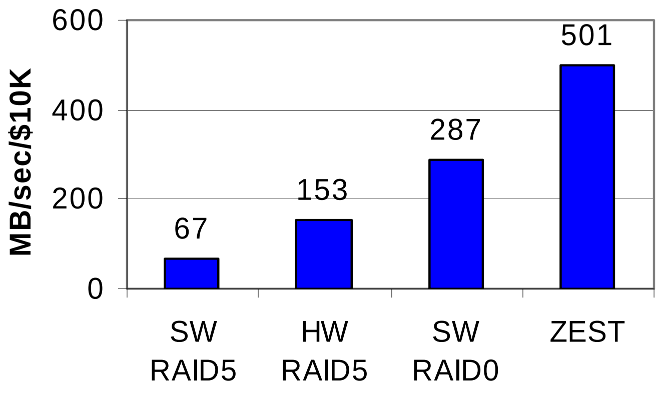
Figure 4a shows Lustre scaling for the RAID0 case -- striping only. Although non-feasible for a production environment, due to a high exposure to risk of lost data, this test gives an upper bound to Lustre's performance for this single-server configuration. Lustre clearly handles thousands of clients, ultimately achieving an average of 430 MB/sec. This shows the added advantage of native "o2ib" routing over InfiniBand. But its performance is more variable, and the best performance is achieved only for the largest stream counts.

Figure 4b shows Lustre scaling for the RAID5 case -- software RAID within the server. Although this configuration would also not be used in production environments, as HPC centers routinely opt for hardware RAID over software RAID, this test demonstrates what is achievable with a modern parallel file system without the support of hardware RAID. In this case, there is a clear penalty to performance when the application exceeds the equivalent of two streams per disk. The resulting bandwidths are unsurprisingly low.

Figure 5a shows the best performance of all of these systems for large stream counts. Efficiency here is defined as the result of dividing the observed aggregate application bandwidth by the aggregate raw spindle speed within the server. For comparison we show an additional data point in this plot corresponding to a production-quality commercial hardware RAID system in use at our center. From this figure it is clearly evident why customers opt for hardware RAID over software RAID. Furthermore we show that Zest achieves a higher efficiency than Lustre, or any other system, even when Lustre dispenses with parity calculations. Recall that Zest not only includes parity calculation at the client, but it also performs multiple checksum calculations throughout the system to ensure the integrity and recoverability of user data for its lifetime within the system.



a)



b)

Figure 5: a) Efficiency and b) write bandwidth per dollar of storage subsystem configurations, as labelled.

Figure 5b shows the aggregate application bandwidth scaled by the cost of the storage subsystems. This is the ultimate measure of the success of any storage system, notwithstanding its usability, scalability and data integrity, which we have already addressed. From this figure we support our final assertion regarding the superior scalability and performance of this storage infrastructure, which is unmatched by any other alternatives.

9.1 Summary of Performance and Features

Here we summarize Zest's measured performance and features, described above.

Scaling

- Our client-server communication protocol supports 1K clients per server.
- There is no central Metadata Service (MDS) in our infrastructure to impose locking between Zest servers. Thus there is no locking contention between Zest servers, so that we expect near perfect scaling with an increase in the number of servers.
- Our client library supports client-side aggregating small writes into larger buffers, so that we expect full performance for 65K writes or larger, and a corresponding decrease of as little as 5% when writing as small as 8K blocks, based on previous measurements.

Performance

- Zest servers have the necessary combination of software, hardware, connectivity and drivers to support writing over at least 4 drives on each of 4 disk shelves at an aggregated efficiency of > 90% of raw spindle speed.
- We have measured backend data rates as high as 1.1 GB/sec out of a possible 1.2 GB/sec aggregated raw spindle speed.
- We have measured aggregated application write bandwidths averaging over 750 MB/sec.
- Zest I/O threads in the server (assigned one thread per disk) are responsible for all I/O to and from each drive. Throughout the benchmarks described above, these threads reported peak write bandwidths of 75 MB/sec – the manufacturer rated performance of the drives.
- Zest maintains a low CPU (<20%) and memory (ingest buffers virtually empty) load during heavy I/O operations.

Usability

- The Zest client intercept transparently supports I/O intensive applications and standard benchmarks as listed, requiring at most recompilation.

10 Conclusions

Zest is designed to facilitate fast parallel I/O for the largest production systems currently conceived (petascale) and it includes features like configurable client-side parity calculation, multi-level checksums, and implicit load-balancing within the server. It requires no hardware RAID controllers, and is capable of using lower cost commodity components (disk shelves filled with SATA drives). We minimize the impact of many client connections, many I/O requests and many file system seeks upon the backend disk performance and in so doing leverage the performance strengths of each layer of the subsystem. We have delivered these features at a per-server efficiency level unmatched within the commercial or academic communities and with an associated performance per price that surpasses all present offerings.

The future of petascale systems rests squarely on highly innovative solutions like this that can optimize performance and deliver intelligent management wherever possible. Equipped with of order 1000 Zest I/O servers such machines will achieve the maximum reliable TBytes/sec/$.

11 Acknowledgements

We are grateful to the developers at Sandia who developed the Portals communication framework for scalable communication and to those at Cluster File Systems who converted that to Lustre LNET, a portable communications library with advanced features like heterogeneous network routing and RPC libraries.

12 References

[1] See http://top500.org/lists/2007/06/performance_development/ for details.
[2] N. T. B. Stone, D. Balog, B. Gill, B. Johanson, J. Marsteller, P. Nowoczynski, D. Porter, R. Reddy, J.R. Scott, D. Simmel, J. Sommerfield, K. Vargo, C. Vizino, "PDIO: High-Performance Remote File I/O for Portals-Enabled Compute Nodes,", *Proceedings of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'06/ISBN #:1-932415-89-0/CSREA), Editor: Hamid R. Arabnia, pp. 925-930, Las Vegas, USA, 2006.
[3] The ASCI Purple "I/O Stress" Benchmark (IOR), documented and available online at http://www.llnl.gov/asci/platforms/purple/rfp/benchmarks/limited/ior/.
[4] The LLNL Scalable I/O Project, documented online at http://www.llnl.gov/icc/lc/siop/.
[5] SPIOBENCH benchmark, description available online at http://www.nsf.gov/pubs/2006/nsf0605/nsf0605.jsp.
[6] NSF Solicitation number 05-0625: documented online at http://www.nsf.gov/pubs/2005/nsf05625/nsf05625.htm.
[7] See http://sourceforge.net/projects/bonnie/ for details and source code.