Zest I/O

Paul Nowoczynski, Jared Yanovich

Advanced Systems, Pittsburgh Supercomputing Center

MSST '08 Baltimore, M

Zest – What is it?

Pittsburgh Supercomputing Center

Parallel I/O system designed to optimize the compute I/O subsystem for checkpointing / application snapshotting.

- Write() focused optimizations – transitory cache with no application read() capability.
- Expose about 90% of the total spindle bandwidth to the application, reliably.
- Emphasizes the use of commodity hardware
- End-to-end design.
  - Client to the disk and everything in between.

Zest: Background

Pittsburgh Supercomputing Center

- Designed and implemented by the PSC Advanced Systems Group (*Nowoczynski, Yanovich, Stone, Sommerfield*).
- Work began in September '06.
- Prototype development took about one year.
- Currently most major features are implemented and in test.

Zest – Why checkpointing?

Pittsburgh Supercomputing Center

Checkpointing is the dominant I/O activity on most HPC systems.

Its characteristics lead to interesting opportunities to for optimization:

- 'N' checkpoint writes for every 1 read.
- Periodic, heavy bursts followed by long latent periods.
- Data does not need to be immediately available for reading

Zest – The impetus.

Pittsburgh Supercomputing Center

Compute performance is greatly outpacing storage system performance.

As a result.. Storage system costs are consuming an increasing percentage of the overall machine budget.

Over the last 7-8 years performance trends have not been in favor of I/O systems

- Memory capacities in the largest machines have increased by~25x
- Disk bandwidth by ~4x

Zest: What can be optimized today?

Pittsburgh Supercomputing Center

Opportunities for optimization in today's parallel I/O systems – do they exist? *YES*

Current systems deliver end-to-end performance which is a ***fraction*** of their ***aggregate spindle bandwidth***.

If this bandwidth could be reclaimed it would mean:

- Fewer storage system components
- Less failures
- Lower maintenance, management, and power costs
- **Improved cost effectiveness for HPC storage systems.**

Zest: Why is spindle efficiency poor?

Pittsburgh Supercomputing Center

Several reasons have been observed:

- Aggregate spindle bandwidth is greater than the bandwidth of the at least one of the connecting busses.
- Parity calculation engine is a bottleneck.
- Sub-optimal LBA request ordering caused by the filesystem and/or the RAID layer.

The first two factors may be rectified with better storage hardware..

The last is the real culprit and is not as easily remedied!

Zest: Software stacks aren't helping.

Pittsburgh Supercomputing Center

Today's storage software architectures (filesystems / raid) generally do not enable disk drives to work in their most efficient mode.

Overly deterministic data placement schemes result in loss of disk efficiency due to seek'ing.

- Pre-determined data placement is the result of inferential metadata models employed by:
  - **Object-based parallel filesystems**
  - **Raid Systems**
- These models are extremely effective at their task but result in data being forced to *specific* regions on *specific* disk drives.
  - ***Results in disk work queues which are not sequentially ordered.***

Zest: Other negative side-effects

Pittsburgh Supercomputing Center

Current data placement schemes complicate performance in degraded scenarios.

In HPC environments, operations are only as fast as the slowest component...

- Object-based metadata and RAID subsystems expect data to be placed in a specific location.
- Difficult or impossible to route write requests around a slow or failed server once I/O has commenced.
- In the current parallel I/O paradigm, these factors have the potential to drastically hurt scalability and performance consistency.

Zest: Methods for optimized writes.

Pittsburgh Supercomputing Center

Zest uses several methods to minimize seeking and optimize write performance.

- Each disk is controlled by single I/O thread.
- Non-deterministic data placement. *(NDDP)*
- Client generated parity.
- No Leased locks

Zest: Disk I/O Thread

Pittsburgh Supercomputing Center

One thread per-disk.

- Exclusive access prevents thrashing.
- Rudimentary scheduler for managing data reconstruction requests, incoming writes, and reclamation activities.
- Maintains free block map
  - Capable of using any data block at any address
  - Facilitates sequential access through ***non-determinism***
- Pulls incoming data blocks from a single or multiple queue called "*Raid Vectors*".

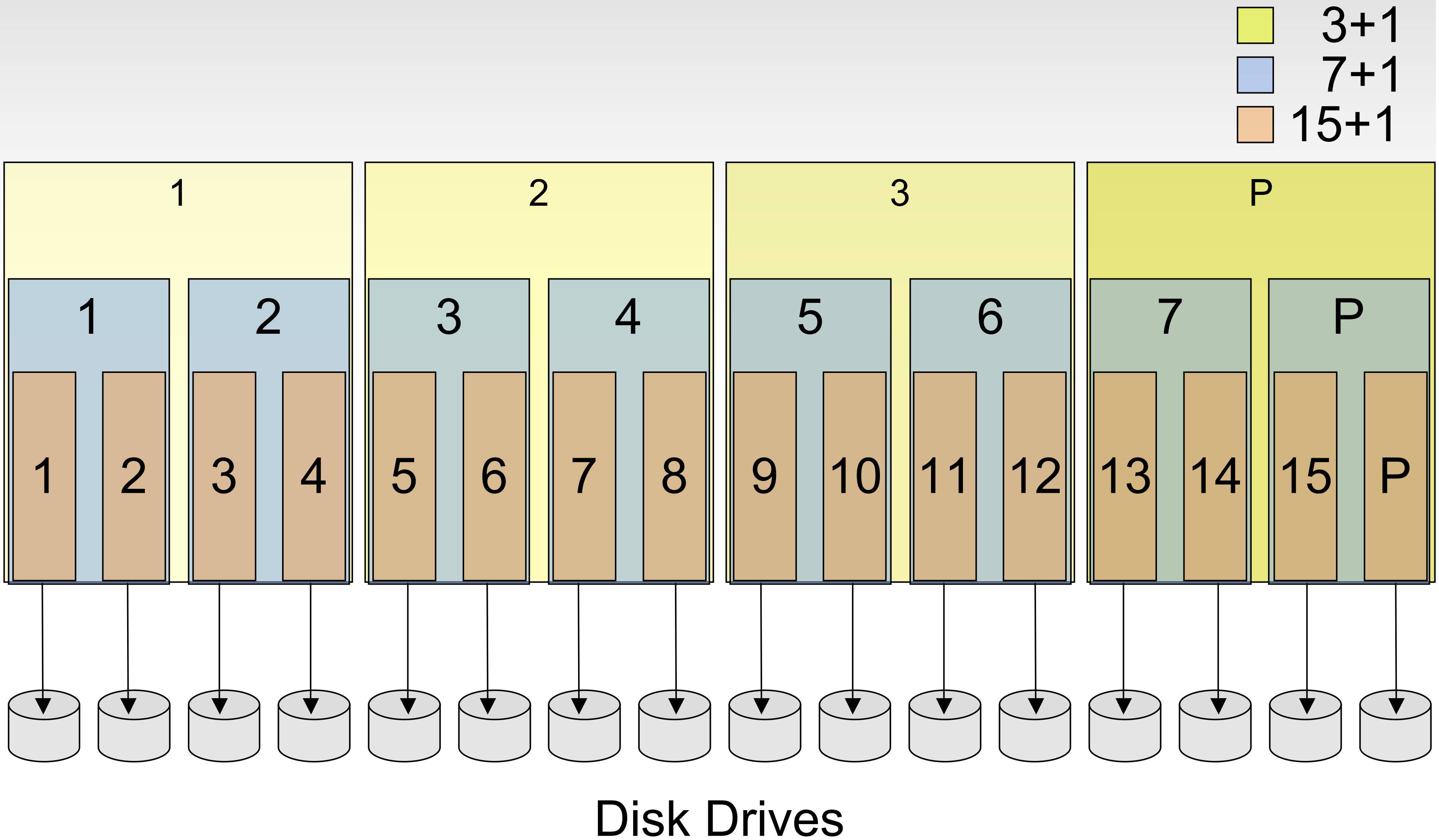
Zest: Raid Vectors

Pittsburgh Supercomputing Center

Queues on which incoming write buffers are placed to be consumed by the disk threads.

- Ensures that blocks of differing parity positions are not placed o the same disk.
- Multiple drives may be assigned to a RV.
  - Blocks are pulled from the queue as the disks are ready.
- Slow devices do less works, failed devices are removed.
- > 1 disk per RV creates a second degree of ***non-determinism.***

Raid Vectors



Zest: Non-deterministic placement

Pittsburgh Supercomputing Center

Non-determinism on many levels:

- Any parity stripe or group may be handled by any ZestION.
  - Slow nodes may be fully or partially bypassed
- Any disk in a Raid Vector may process any block on that vector.
  - Assumes that ndisks > (2 x raid stripe width)
- Disk I/O thread may place data block at the location of his choosi
  - Encourages sequential I/O patterns.

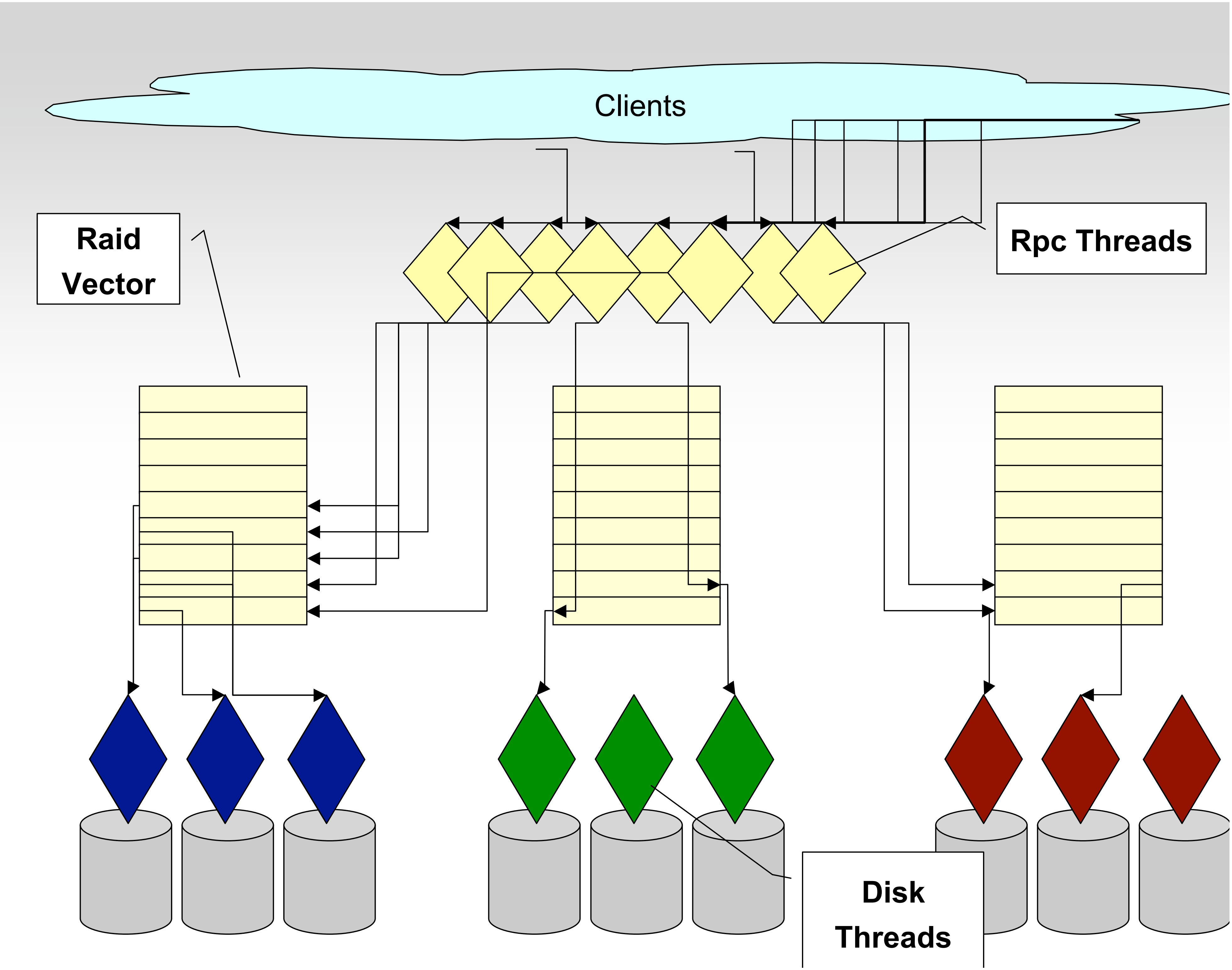
Performance is not negatively impacted by the number of clients the degree of randomization within the incoming data streams

Zest: Client Parity, CRC, and Cache

Pittsburgh Supercomputing Center

Much of the hard work is placed onto the client preventing th ZestION from being a bottleneck.

- Data blocks are Crc'd and later verified by the ZestION during the post-processing phase.
- Data verification can be accomplished without read back of the entire parity group.
- Client computed parity eliminates the need for backend raid controllers.
- Client caches are not page based but vector-based.
  - No global page locks needed.
  - Further eliminates server overhead and complexity.


Clients
Raid
Vector
Rpc Threads
Disk
Threads

Zest: NDDP – the cost..

Pittsburgh Supercomputing Center

Increasing entropy allows for more flexibility but more bookkeeping is required.

NDDP destroys two inferential systems, one we care about th other is not as critical (right now).

- Block level Raid is no longer semantically relevant.
- Tracking extents, globally, would be expensive.

Zest: NDDP – the cost..

Pittsburgh Supercomputing Center

Declustered Parity Groups

- Parity group membership can no longer be inferred.
- Data and parity blocks are tagged with unique identifiers that prove their association.
  - ✓ Important for determining status upon system reboot.
- Parity group state is maintained on separate device.
  - ✓ Lookups are down with diskID, blockID pair.

Zest: NDDP – the cost..

Pittsburgh Supercomputing Center

File Extent Management

Object-based parallel file systems (i.e. Lustre) use file-object maps to describe the location of a file's data.

- Map is composed of the number of stripes, the stride, and the starting stripe.
- Given this map, the location of any file offset may be computed.

Zest has no such construct!

- Providing native read support would require the tracking of a file offset, length pairs.
- Extent storage is parallelizable.

Zest: NDDP – additional benenfits.

Pittsburgh Supercomputing Center

Since any parity group may be written to any I/O server:

- Failure of a single I/O server does not create a hot-spot in the storage network.
  - ✓ Requests bound for the failed node may be evenly redistributed tc the remaining nodes.
- Checkpoint bandwidth partitioning on a per-job basis is possibl

Zest: Post-processing

Pittsburgh Supercomputing Center

Begins once the data ingest phase has halted or slowed.

- Current post-processing technique rewrites the data into a lustre filesystem. (*syncing*)
- In the future, other data processing routines could make use of the same internal infrastructure..

Zest: Post-processing / Syncing

Pittsburgh Supercomputing Center

How does Zest sync file data?

- Zest files are 'objects' identified by their Lustre inode number.
  - These are hardlinked to their lustre equivalents on create().
- On write() the client:
  - The data buffer
  - Metadata slab containing:
    - Inode number, Crc, Extent list, etc.
- Syncing is done using the hardlinked immutable path, the inode, and the extent list.

Zest: Reliability

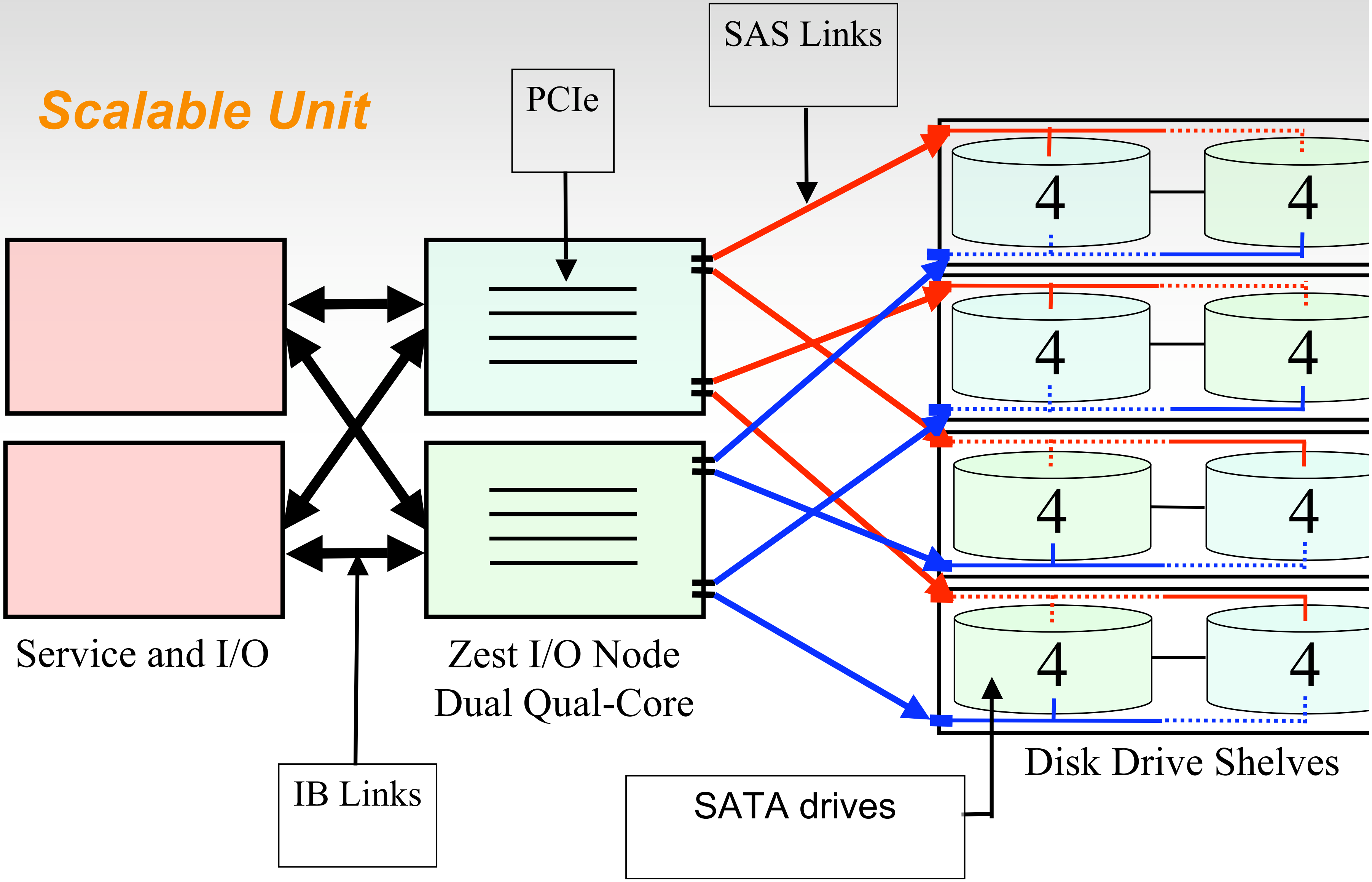
Pittsburgh Supercomputing Center

Zest provides reliability on par with a typical HPC I/O system.

- Data redundancy through Raid.
- Recoverability via multi-homed disk configuration.

Zest supports hardware configurations such as the following.

Scalable Unit



- No single point of failure

Zest: Reliability Features

Pittsburgh Supercomputing Center

- Support for failover pairs.
  - Zest superblocks are tagged with UUIDs to avoid confusion in shared disk configurations.
- On reboot, corrupt or missing data is rebuilt, unsynchronized data is rectified.
- Certain modes of disk failure are easily detected and the I/(
thread is quarantined.
- 'Fast rebuild' is supported.
  - When a disk fails, the Zest server has an list, in memory, of all the acti
blocks.   Those blocks can rebuilt immediately without scanning the entire set.

Zest: Performance Result

Pittsburgh Supercomputing Center

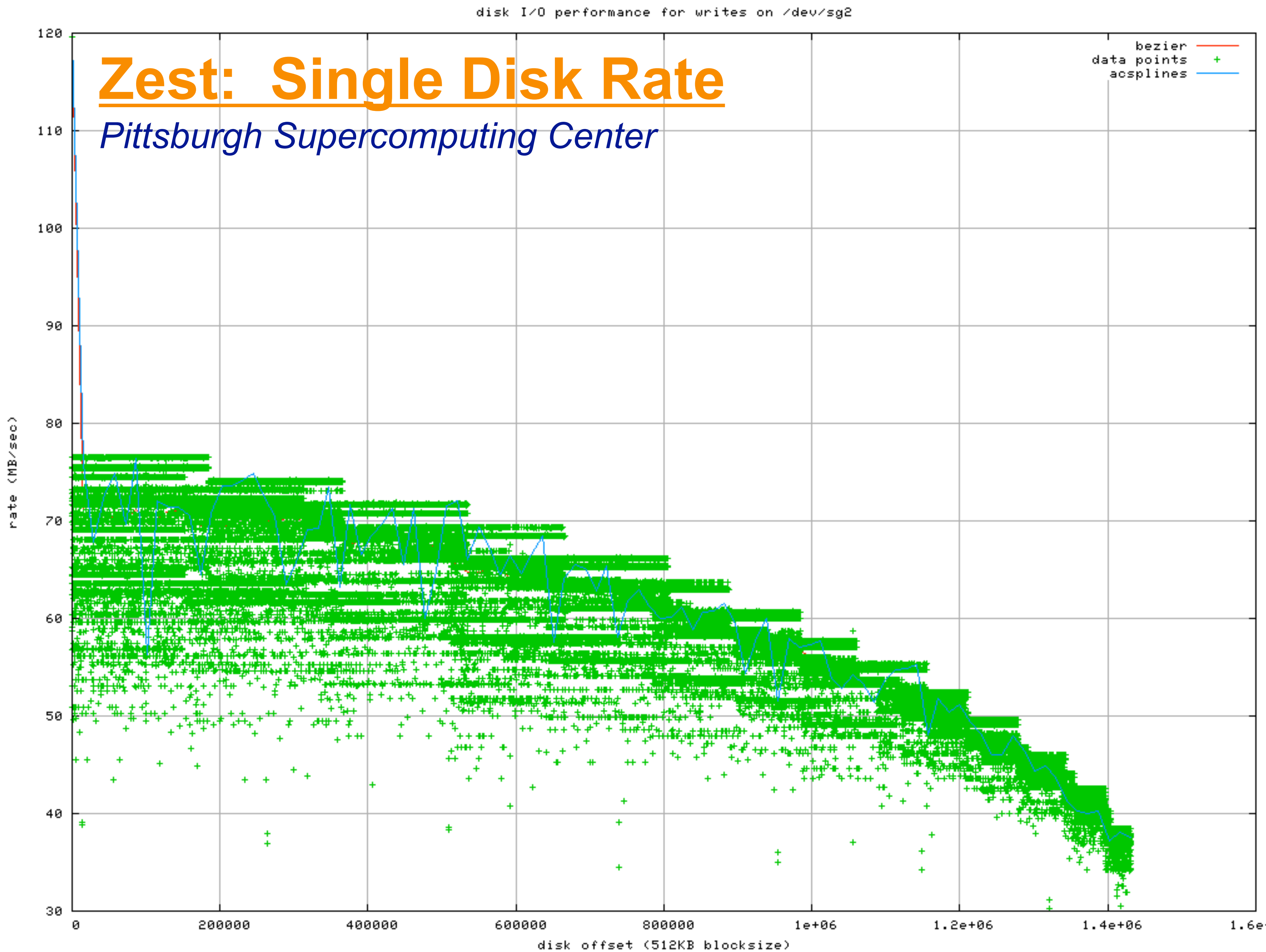
- Test consisted of sequentially writing from each PE into a separate file.
- Clients used a 7+1 Raid5 parity scheme (12.5% overhead)

Zest Server Hardware

- ✓ 2 x 4 Core Intel Processors
- ✓ Multiple PCI-e Busses
- ✓ 1 Sas Controllers
- ✓ 1 IB Interface (DDR)
- ✓ 12 Drives (@75MB/s per)

Zest: Single Disk Rate

Pittsburgh Supercomputing Center



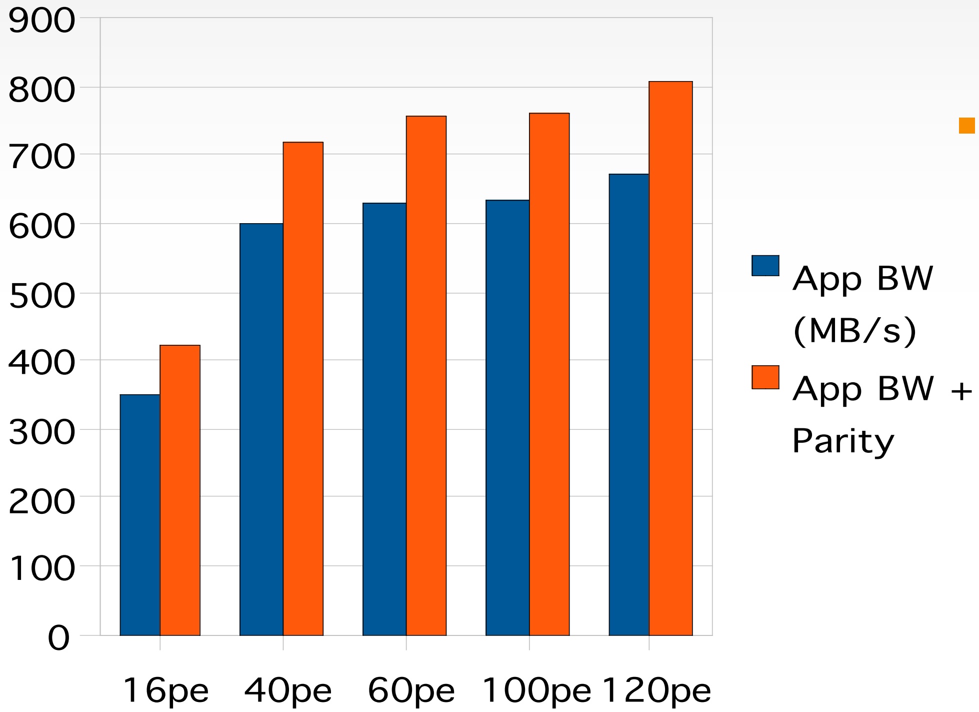
Zest: Performance Result

Pittsburgh Supercomputing Center

By itself, the Zest backend can easily reach 90% efficiency.

- 12 disks@860MB/s
- Very low CPU utilization due to zero-copy and scsi generic I/O (sg)
  - About 5% of 8 cores.

Zest Performance – Linux cluster



- Best case (120pe's), application sa
  75% of spindle bandwidth.
- If parity overhead is ignored the transfer rate represents 89.6% of th
  spindle bandwidth!