Birds of a Feather

SC2004

Pittsburgh, PA

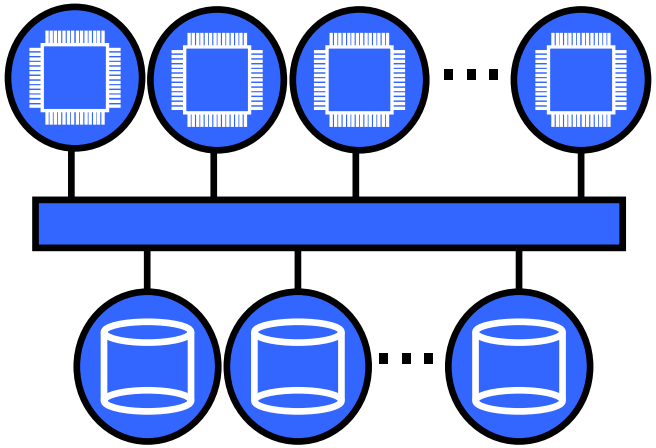
http://www.pvfs.org/pvfs2

Outline of presentation

- Introduction (Walt)
- PVFS2 design features (Neill)
- MPI-IO and PVFS2 (RobL)
- PVFS2 present and future (RobR)

- Open discussion

I/O in a HPC system



Clients running applications
(100s-10,000s)

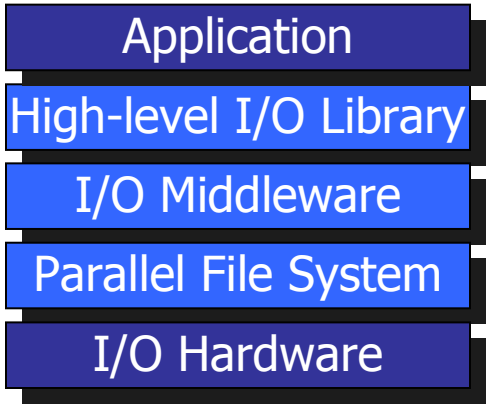
Storage or System Network

I/O devices or servers
(10s-1000s)

- HPC applications increasingly rely on I/O subsystems
  - Large input datasets, checkpointing, visualization
- Programmers desire interfaces that match their problem domain
  - Multidimensional arrays, typed data, portable formats
- Two issues to be resolved by I/O system
  - Very high performance requirements (concurrent access to HW)
  - Gap between app. abstractions and HW abstractions
- Software required to address <u>both</u> of these problems

I/O software stacks

- Computational science applications have complex I/O needs
  - Performance and scalability requirements
  - Usability (Interfaces!)
- Software layers combine to provide functionality
  - High-level I/O libraries provide useful interfaces
    - Examples: Parallel netCDF, HDF5
  - Middleware optimizes and matches to file system
    - Example: MPI-IO
  - Parallel file system organizes hardware and actually moves data
    - Examples: PVFS1, PVFS2, GPFS, Lustre



Role of parallel file systems

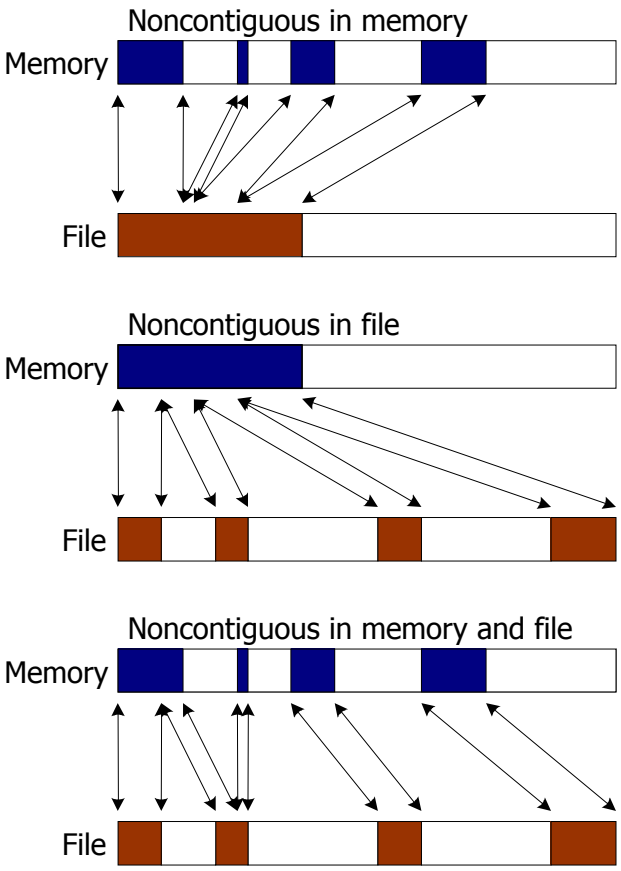
- Manage storage hardware
  - Lots of independent components
  - Presenting a single logical view
  - Providing data redundancy, maybe
- Scale to very large numbers of clients
  - Handling many concurrent, independent accesses
  - Considering client failures to be a common case
- Provide building-block API and semantics
  - Usable by I/O middleware (MPI-IO)
  - Remembering importance of efficiency
- Recognize that this is only one piece of the I/O system

Existing solutions, problems

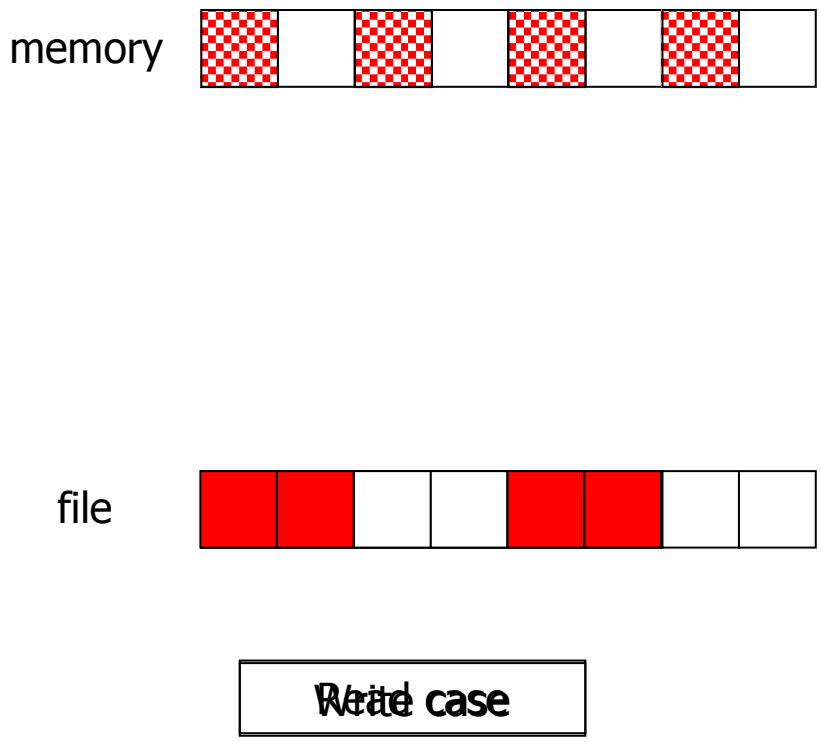
- Interfaces are inadequate for efficient access
  - Noncontiguous access
  - Primitives for scalable I/O (including metadata)
- Consistency semantics aren't "just right"
  - POSIX scope is too big; difficult to implement with performance and reliability
  - NFS guarantees are too small; not enough guaranteed to be useful for parallel applications
- Architectures are too complicated to manage
  - Locking systems
  - Fault tolerance

I/O Capabilities

- Noncontiguous I/O operations are common in computational science applications
- Most PFSs available today implement a POSIX-like interface (open, write, close)
- POSIX noncontiguous support is poor:
  - readv/writev only good for noncontiguous in memory
  - POSIX listio requires matching sizes in memory and file
- Better interfaces allow for better scalability



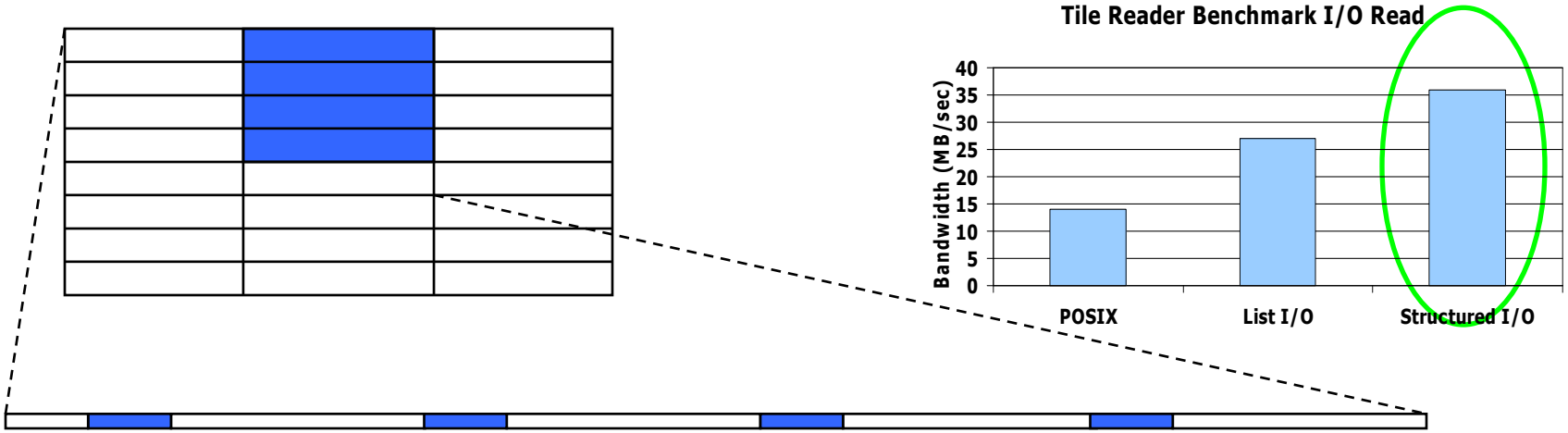
List I/O interface



- A single I/O operation can handle entire noncontiguous I/O access pattern
  - If PFS supports this API
  - Available in PVFS1
- Overhead in passing file offsets and lengths across network on an I/O operation
- Description of I/O can be bigger than data!

Even better: Structured I/O

- Often there is regularity in noncontiguous accesses
- We can use regularity to more efficiently express I/O operation
  - Just like we do with MPI datatypes: vectors, indexed types
- Ideal building block for scientific applications



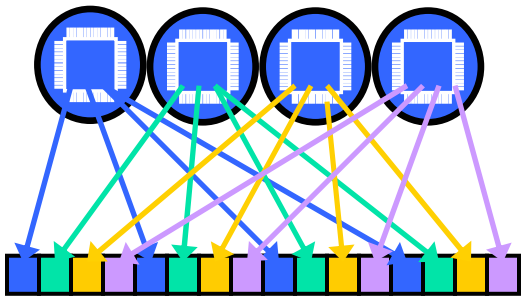
Results from "Datatype I/O" prototype in PVFS1

Consistency semantics

- Most PFSs implement something similar to POSIX consistency semantics
  - Reads and writes are atomic with respect to <u>all other processes</u> that might access the file system (sequential consistency)
  - Interface gives PFS little information
  - Very difficult to implement with high performance
    - Must somehow be tracking all those processes
- Alternative, NFS, is too loose
  - Behavior for concurrent access undefined!

Consistency in parallel apps

- Split the responsibility
  - MPI-IO layer manages concurrent access by group of processes
  - PFS just provides basic building block: atomic non-overlapping writes
    - Call this "nonconflicting write consistency"
    - Sufficient for majority of applications
    - Higher performance, simpler to implement
- Other consistency semantics (e.g. close-to-open) might be right for home directories
  - NFS consistency considered sufficient in this role



With nonconflicting write consistency, writes to non-overlapping regions occurring in any order, are guaranteed to be visible in the file, to all subsequent reads, after the write completes.

Note: Must work across block boundaries!

Complexity and fault tolerance

- Fault tolerance is easy when there are no dependencies between components
  - Nothing to get out of synchronization!
  - Those dependencies usually take the form of shared state – data that exists in more than one place
    - Example: disk blocks cached in memory
- Minimizing shared state should be a priority when designing a fault tolerant parallel file system

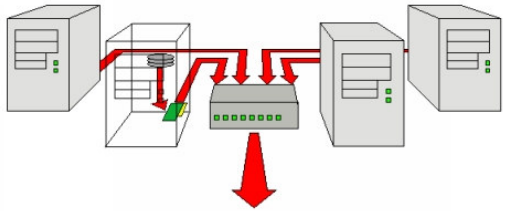
Locking subsystems

- Applied to file data to provide POSIX consistency semantics
  - When someone wants to do I/O, they must first get a lock
- Locks are shared state
  - If a client dies with a lock, somehow it must be reclaimed
  - Because of that shared state, lock servers are difficult to replicate for failover
- Locks and fault tolerance don't mix well

Lockless file systems

- We can implement PFSs without locks
- Change the file I/O consistency semantics
  - Implement nonconflicting write consistency as building block
  - Let MPI-IO handle remaining consistency issues for concurrent access
- Provide atomic metadata operations
  - So file creates, removes, and renames maintain consistent name space
- Resulting system is simpler, more tolerant of failures, and better tuned for MPI-IO

Our first effort: PVFS1



- Started at Clemson University in mid 1990s
- First designed as a research tool
  - Examine some of the issues we just described
    - I/O interface capabilities
    - Consistency semantics
- Eventually released and used as high-performance scratch space
  - Current version is 1.6.3
  - Continues to be supported
- A lot was learned from this project
- Time is right for a new parallel file system

A fresh start: PVFS2

- All new parallel file system implementation
  - Kept only a familiar name, philosophy
- Architected for easy modification
- Accessible to a wide audience
- Designed for scientific applications on very large systems
  - Structured I/O, scalable metadata operations
  - Nonconflicting write consistency
  - Stateless, lockless system
  - Tolerant of client failures, amenable to commodity fault tolerance solutions

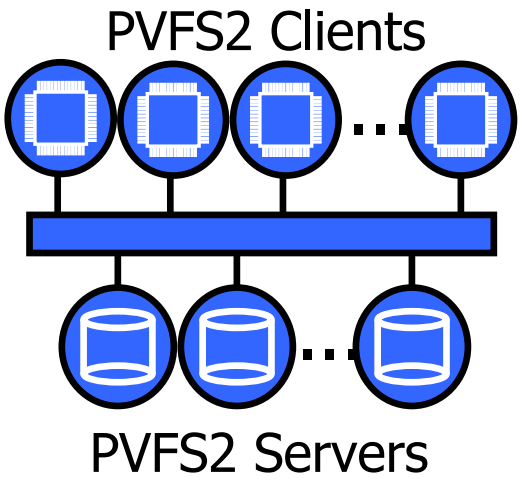
Multi-institution collaboration

- PVFS2 is an open, collaborative effort
- Core development
  - Argonne National Laboratory
    - Ross, Miller, Latham, Gropp, Thakur
  - Clemson University
    - Ligon, Carns (defending!), Settlemyer
  - Ohio Supercomputer Center
    - Wyckoff, Baer
- Collaborators
  - Northwestern University
    - Choudhary, Ching
  - Ohio State University
    - Panda, Wu (graduated!)
  - Penn State University
    - Sivasubramaniam, Kandemir, Vilayannur

PVFS2 Design Features

http://www.pvfs.org/pvfs2

PVFS2 at a glance

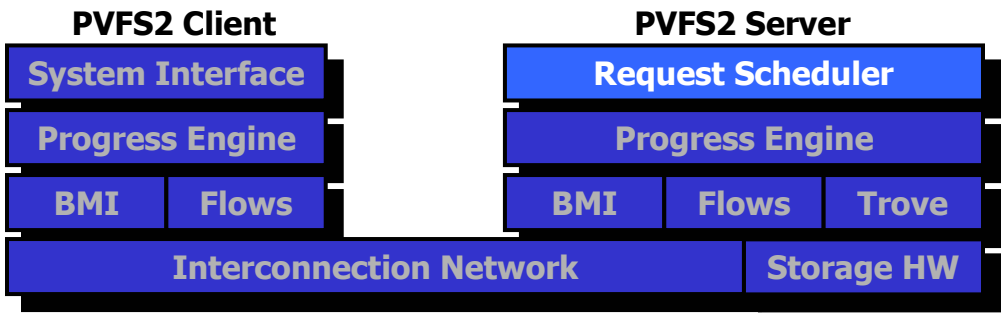


- "Intelligent" servers with PFS-oriented protocol
  - Single server type
  - Can manage metadata, data, or both
  - File data distributed across many servers
- Messaging over existing communication network
  - Independent servers communicate only with clients
  - Leverage that expensive network
- Storage on disks locally attached to servers
  - Possibly shared for failover purposes
  - Both data and metadata may be distributed across servers
- MPI-IO and VFS interfaces for clients

Scalability

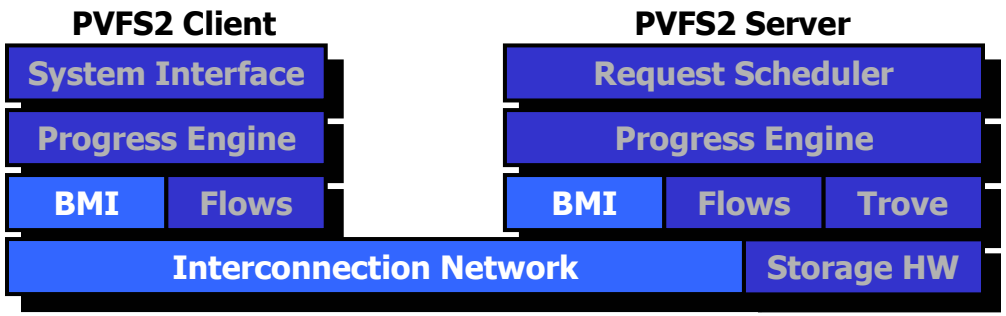
- Clients are independent
  - No locking system to tie them together
- No need to contact metadata server on every I/O operation
  - Deterministic mapping of data placement on servers (per file)
- Servers are threaded to handle concurrent operations
  - Better utilization of network and storage
- Nonconflicting write consistency semantics allow for maximum I/O concurrency to a single, shared file

Scheduling operations



- Request scheduler on each server allows fine-grained control over ordering of operations
  - Permits concurrent I/O where valid
    - Enforces nonconflicting write semantic
  - Ensures atomic metadata reads and updates

Support for multiple networks



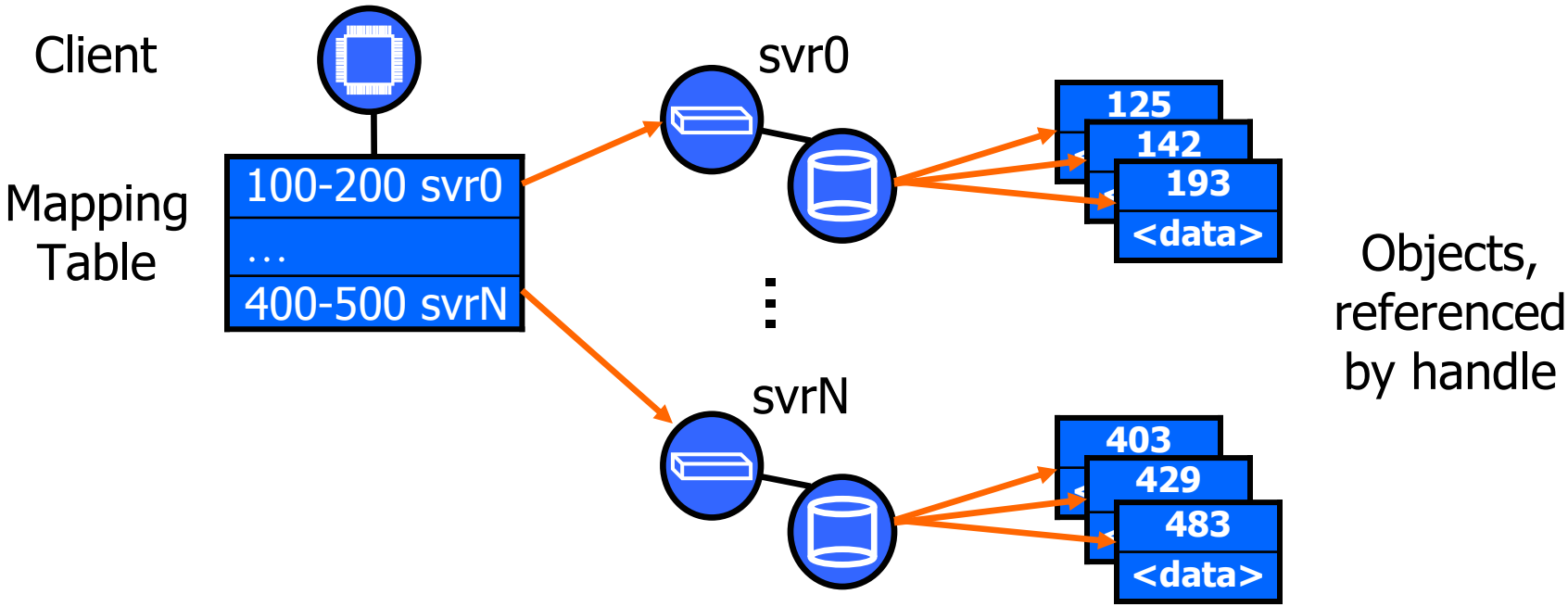
- Abstract BMI interface hides network details
- Native support for
  - TCP/IP
  - Myrinet GM
  - InfiniBand
- Multi-homed servers supported
- Messages encoded for heterogeneous systems
- BMI implementation on Quadrics Elan4 in progress

Distributed metadata

- For applications that access many small files, distributing metadata eliminates an additional bottleneck
- PVFS2 can be configured to place metadata on any number of servers
- Replication of metadata is a separate issue

Mapping files to servers

- Files made up of objects stored on various servers
- Client-side configuration data provides a mapping from object handles to servers
- Eliminates searches and additional indirection
  - Minimizes latency



Fault tolerance

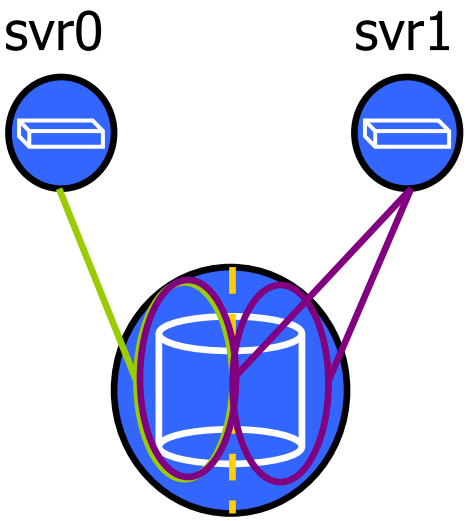
- PVFS2 is a stateless system
  - Servers operate as independent entities
  - Clients are independent with respect to one another
  - Clients do not cache data that is necessary for correct operation
  - No locking system is employed
- This simplifies the handling of both client and server failures

Tolerating client failures

- In very large systems, client failures are likely to be common
- The PVFS2 design allows client failures to be ignored completely by servers and other clients
  - Clients are not responsible for any file system data
  - As opposed to file systems using locks, where locks and dirty blocks must be recovered

Failover for servers

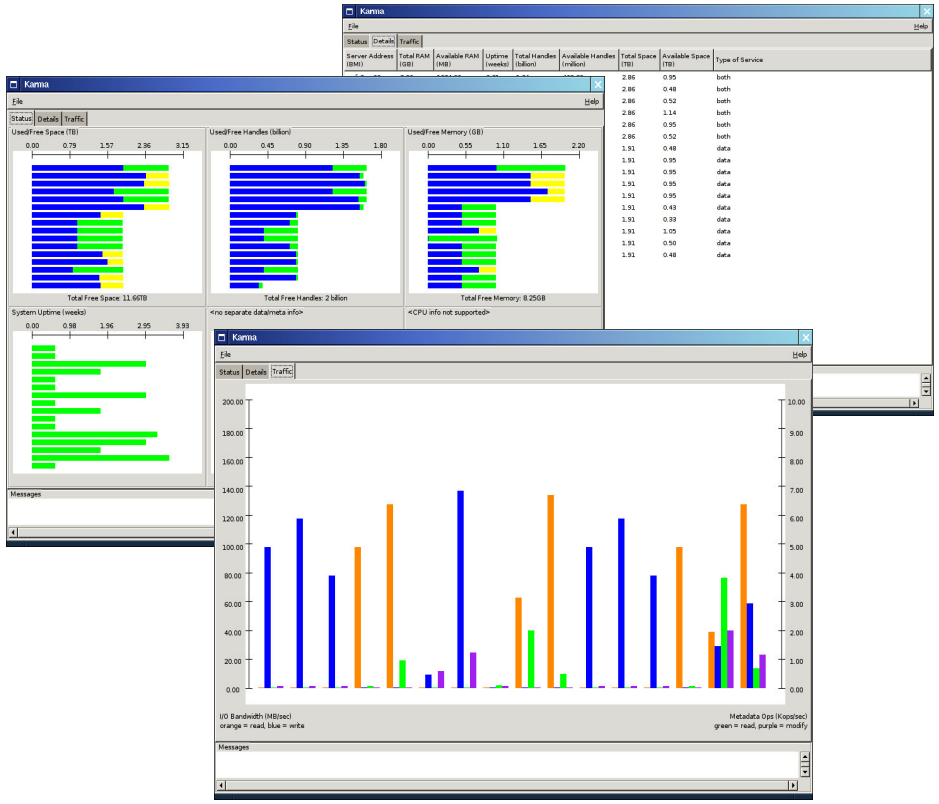
- Likewise, the PVFS2 server design fits well with commodity failover solutions
  - Paired servers share a partitioned storage device, each backing up the other
    - Active/Active mode
  - When one server fails, other server takes over
  - Because there is no shared state, clients simply reconnect



Shared storage

Accessibility and management

- Doesn't take a rocket scientist to install
  - One server type, simple configuration
- Can be re-exported via NFS for access on other OSes
- GUI and command line tools for monitoring and administration
  - pvfs2-fsck for salvaging damaged file systems
  - Migration tools, etc. planned but not yet in place



Linux VFS support

- Lightweight kernel module and user-space client allow Linux nodes to mount PVFS2 file systems
- Supports all the usual stuff, plus
  - Private mmap() reads and execution
  - Symlinks
- Linux 2.4 and 2.6 kernels on IA32, IA64, Opteron, PowerPC, Alpha
  - Including mixes of these in the same file system

MPI-IO and PVFS2

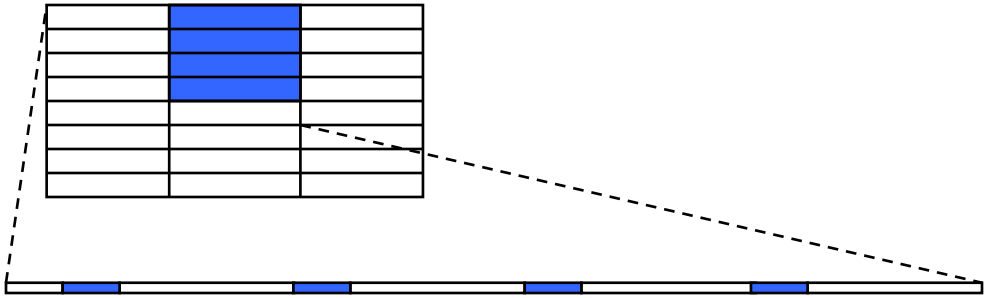
http://www.pvfs.org/pvfs2

MPI-IO Interface

- One goal of the MPI-IO implementer is to eliminate the linear relationship between number of processes and number of file system operations
- Implementation options depend on characteristics of the underlying file system API
  - How can we "talk" to the file system?
    - Richness of the API
  - Who else knows what was said?
    - Consistency semantics
- Two categories of operations:
  - I/O operations (e.g. MPI_File_read_at)
  - Management operations (e.g. MPI_File_delete)
- ROMIO MPI-IO implementation provides many optimizations that leverage PVFS2 features

Noncontiguous I/O

- File views and MPI datatypes allow users and libraries to describe noncontiguous I/O



- PVFS2 allows these regions to be described concisely as well
- ROMIO MPI-IO implementation can convert MPI-IO descriptions into PVFS2 ones
  - Not optimal yet; some performance tuning left

Scalable management operations

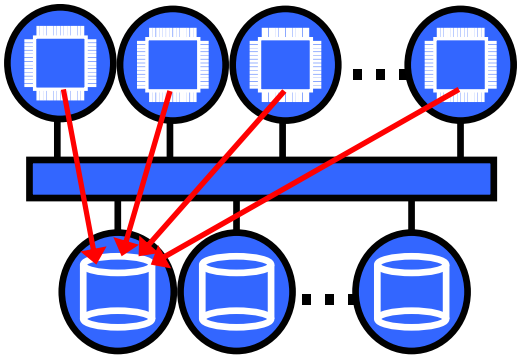
- Most are collective, providing opportunity for optimization
- File system interfaces and semantics also affect management operations
- File access model (e.g. stateful file descriptors) impacts scalability of MPI-IO open and close
- Name space consistency impacts operations such as MPI-IO delete
- Cache consistency and policy impacts operations such as MPI-IO sync and resize

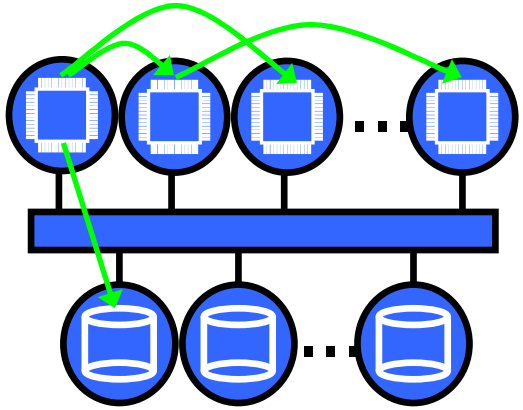
Management operations

Function	Collective?	Number of File System Operations		
		NFS	POSIX	PVFS2
MPI_File_get_size	No	O(n)	O(n)	O(n)
MPI_File_seek	No	O(n)*	O(n)*	O(1)*
MPI_File_delete	No	O(1)	O(1)	O(1)
MPI_File_open	Yes	O(n)	O(n)	O(1)
MPI_File_close	Yes	O(n)	O(n)	O(1)
MPI_File_sync	Yes	O(n)	O(n)	O(1)
MPI_File_set_size	Yes	O(n)	O(1)	O(1)
MPI_File_preallocate	Yes	O(1)	O(1)	O(1)
MPI_File_set_info	Yes	O(n)	O(n)	O(1)
MPI_File_set_view	Yes	O(n)	O(n)	O(1)

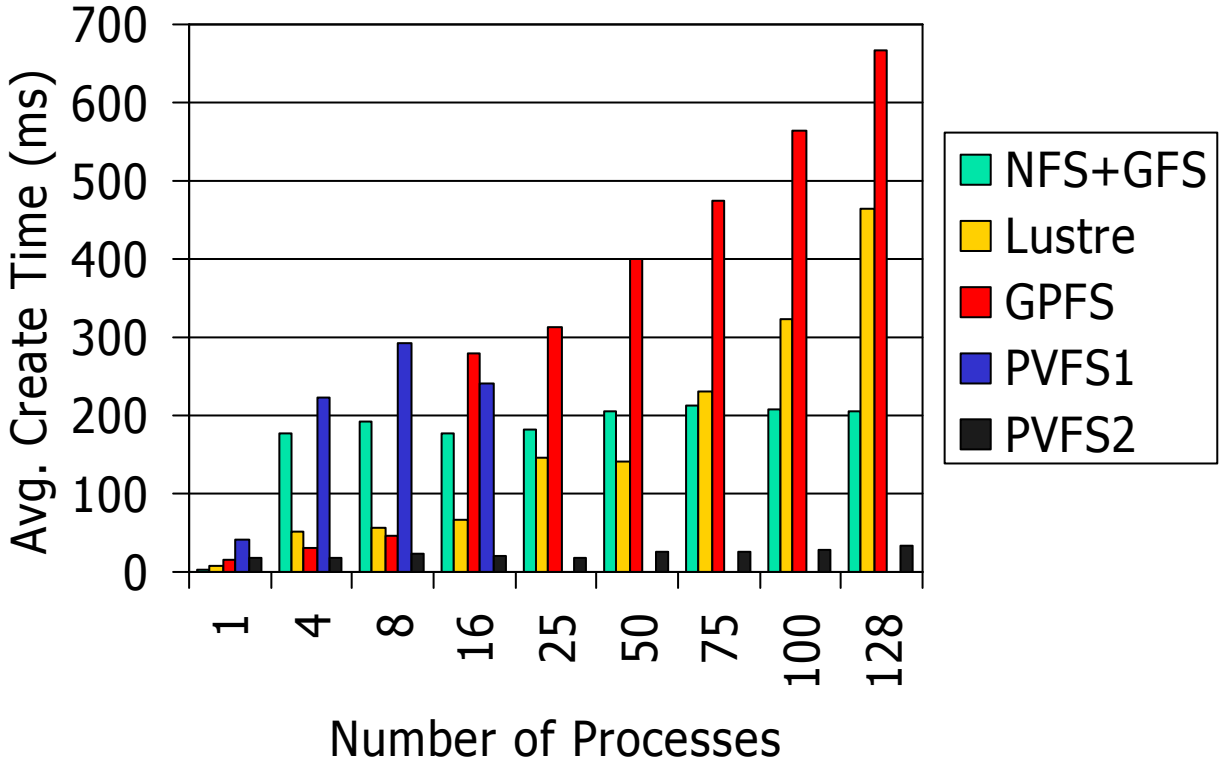
Scalable I/O: MPI_File_open

- POSIX file descriptor model limits scalability of MPI-IO operations
  - Forces all processes to make open and close calls
  - 2000 opens = system call storm!!!
- PVFS2 uses handle system instead
  - One process performs file name resolution
  - MPI_Bcasts the result to other processes
  - File system traffic independent of number of processes!
- Other operations are similar
  - close, sync, truncate, etc.





MPI-IO file create scalability



Notes: Smaller is better!

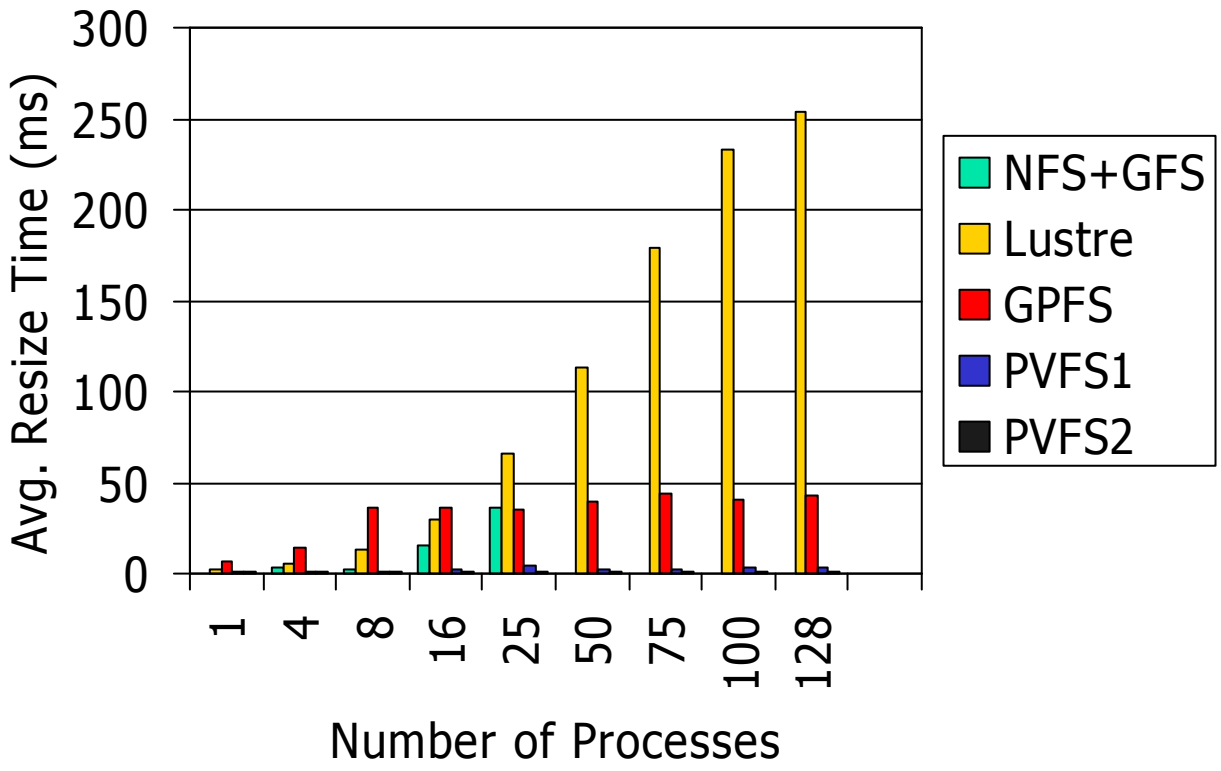
PVFS1 time skyrockets at 25 clients (2.15 sec); data omitted.

Scalable MPI_File_set_size

- Resizing a file can be performed scalably when all nodes can immediately see file size changes
  - Ok for true POSIX, PVFS, PVFS2
  - Not guaranteed for NFS

```
if (rank == 0) {
   /* truncate on one node only */
   ret = ftruncate(fd, size);
   MPI_Bcast(&ret, 1, MPI_INT, 0, comm);
}
else {
   /* Bcast the result to others */
   MPI_Bcast(&ret, 1, MPI_INT, 0, comm);
}
```

MPI_File_set_size scalability



Notes: Smaller is better!

NFS+GFS time skyrockets at 50 clients (1.96 sec); data omitted.

Summary of MPI-IO on PVFS2

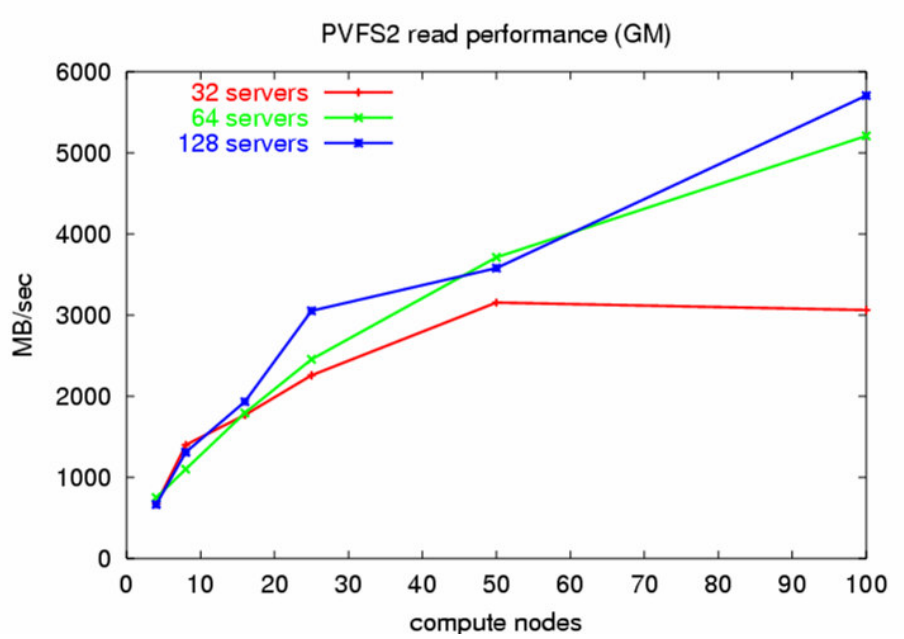
- Our ability as MPI-IO implementers to create scalable solutions depends on the underlying file system
  - Descriptiveness of API
  - Consistency semantics
- File system interfaces can be tailored to better support scalable MPI-IO
- PVFS2 was designed with MPI-IO in mind
  - Almost all MPI-IO operations, I/O and management, are scalable on PVFS2
  - As machines continue to grow in node count, this will be ever more important

PVFS2 Present and Future

http://www.pvfs.org/pvfs2

PVFS2 status

- Testing with hundreds of clients, hundreds of servers, as available
  - Both "easy" access patterns and more difficult ones
- Much of the system is in place
  - IA32, IA64, PPC, Alpha Linux
    - Linux 2.4 and 2.6 kernels
  - Networks: TCP, Myrinet, IB
  - Storage on local file systems
    - Data in UNIX files
    - Metadata in Berkeley DB
  - Distributed (not replicated) metadata
  - Failover
  - Basic monitoring GUI
- 1.0 release is out!



Data obtained using ANL LCRC Jazz machine with file sizes that would fit in cache on Jazz nodes (Jazz nodes have single, relatively slow disks).

We'd be happy to test on a larger system if someone wants to let us borrow one ☺.

Comparison of features

File System	Version	Linux 2.4	Linux 2.6	Stock Kernel Support	Native Myrinet	Quadrics	Infini Band	Distrib. Metadata	Opt. for MPI-IO	Failover
PVFS2	1.0	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes
PVFS1	1.6.3	Yes		Yes					Yes	
Lustre	1.0.4	Yes				Yes				Yes
GFS	6.0	Yes						Yes		Yes

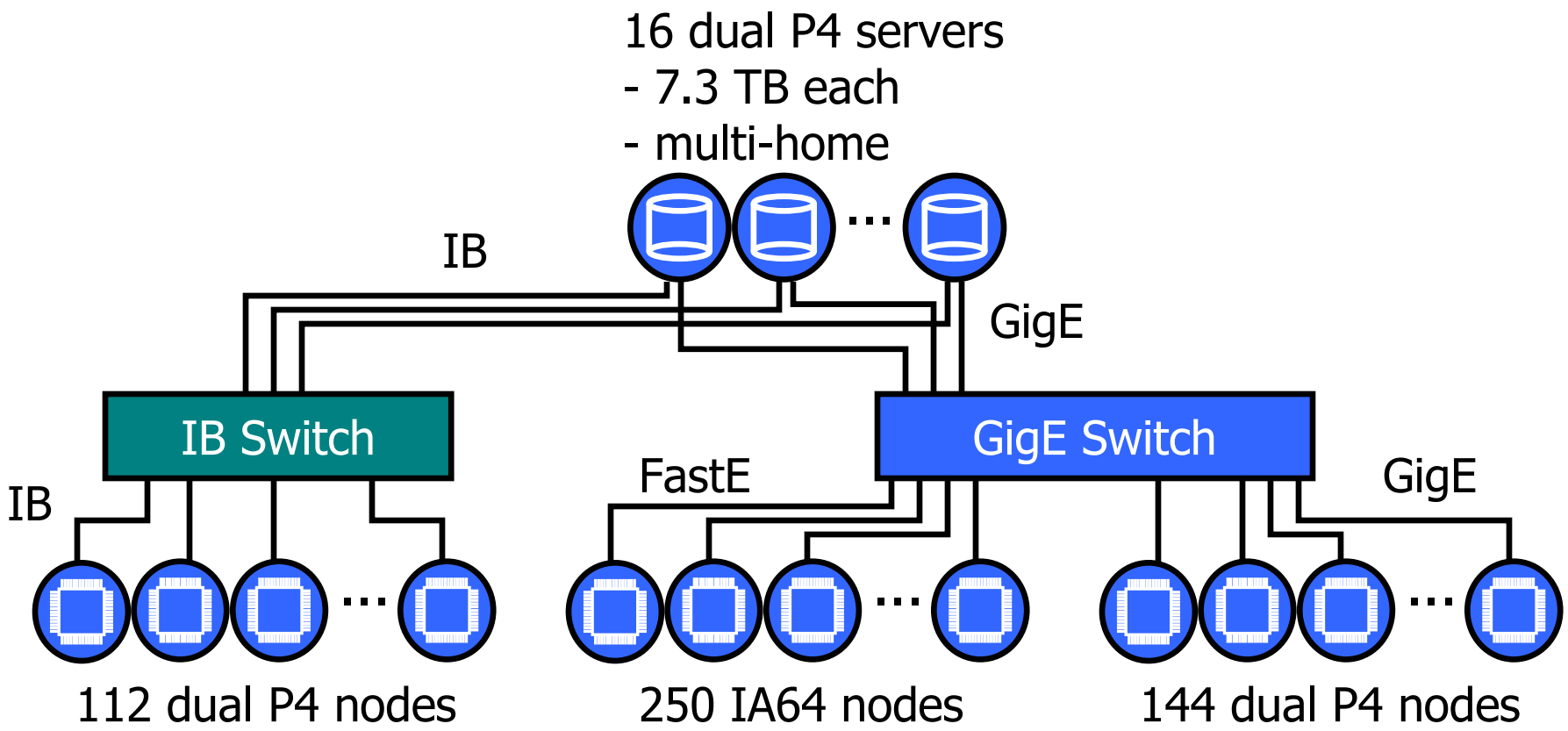
- Comparing against most up-to-date, freely available versions
- References:
  - http://www.clusterfs.com/compare.html
  - http://www.redhat.com/docs/manuals/csgfs/

Recommended configurations

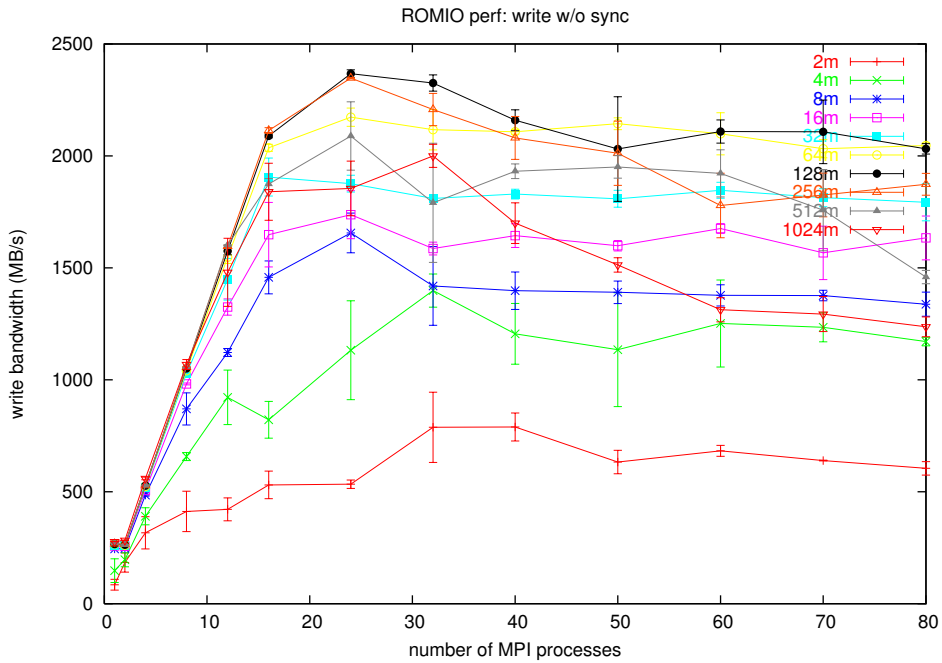
- Dedicated servers
  - Eliminates contention for system resources
  - More consistent performance when more than one application is running
- Linux 2.6.0+
  - O(1) scheduler and NTPL support improve performance
- glibc 2.3.2+
  - Earlier versions have broken AIO callbacks, and the workaround imposes a performance hit
- gcc 3.0+
- ext3 with writeback data mode
  - Fast fsck, widely supported and used, good performance
- Berkeley DB 3.3+
  - Newer versions are more flexible WRT synchronization

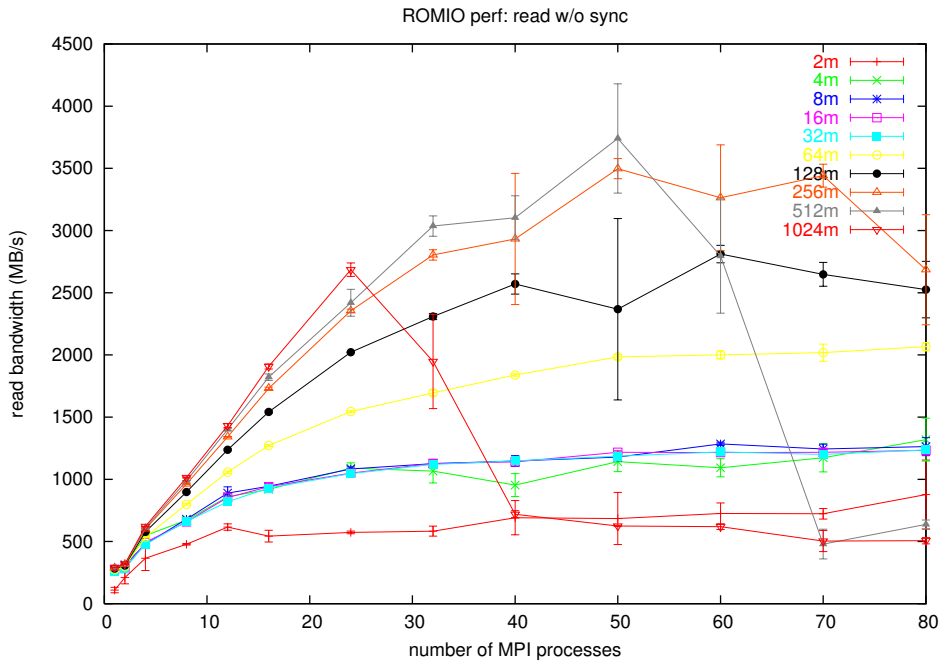
PVFS2 at OSC

- 506 total clients
- 116.8 TByte file system



OSC cluster performance





- Data sizes are per-client
- Achieving ~2.8GB/sec write, ~3.8GB/sec read
  - No network optimization (memory registration or pipelining)

PVFS2 near future

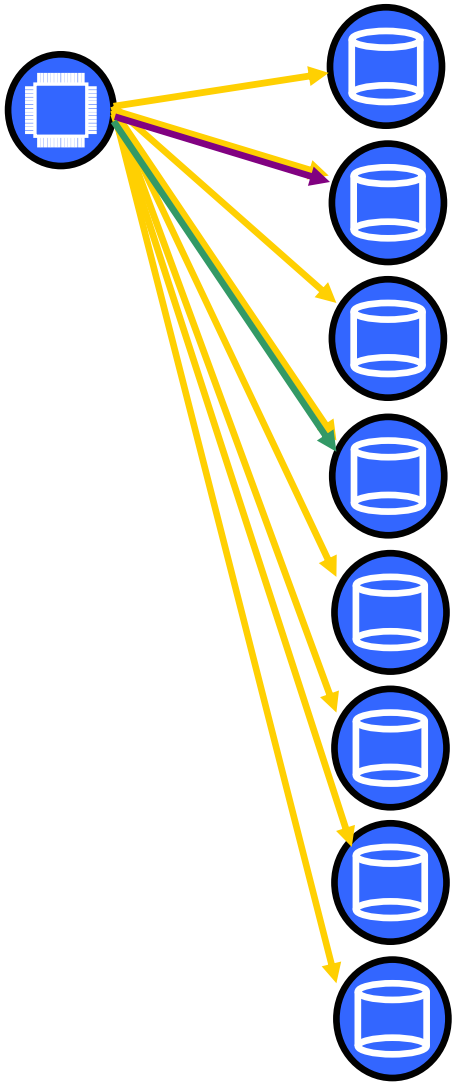
- Performance tuning
  - Improved use of PVFS2 types in ROMIO
  - Elimination/minimization of metadata operations during I/O
  - Adjusting BMI and Flow code based on performance testing
- Support and bug fixing
  - With the 1.0 release, at least some of you will try this out (we hope!)
  - New users bring new questions and new bugs

PVFS2 future, continued

- Even more efficient operations
  - Structured, intra-server collective communication
- Alternative, lower-cost redundancy schemes
  - Avoiding expensive hardware through new algorithms/approaches
  - User-directed redundancy
- User-supplied hints for optimization
  - Immutable files
- Extended attributes
- Support for very large scale systems

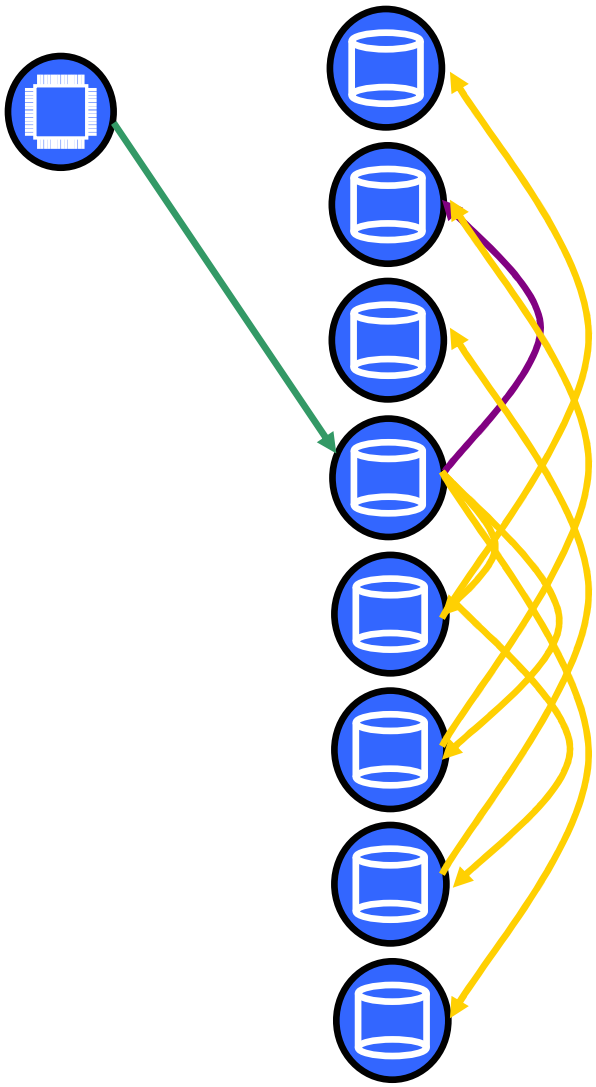
File system operations

- Many file system operations require a sequence of steps to complete
- Example: File creation
  - Create metadata object
  - Create objects to hold data on N servers
  - Create new directory entry
- Currently clients perform these steps
  - Some serialization occurs
  - Failure while in progress leaves orphaned objects
  - High latency if client is far from servers



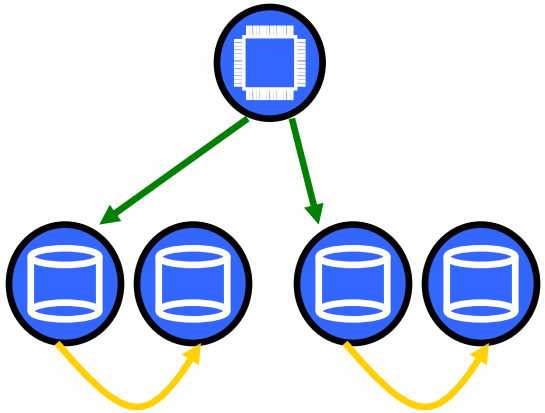
Inter-server communication

- We can move these steps into the server pool instead
  - Servers have storage, can log in-progress operations
  - Single coordination point for some operations (e.g. create)
  - Servers tend to be near each other (lower latency)
  - Servers can organize communication similar to collective communication
    - Directory entry server gets message from client
    - Creates objects to hold data using tree-based communication
    - Creates metadata object, returns ACK to client
- No changes to file layout or fault tolerance characteristics
- Work by Phil Carns, Clemson Univ.

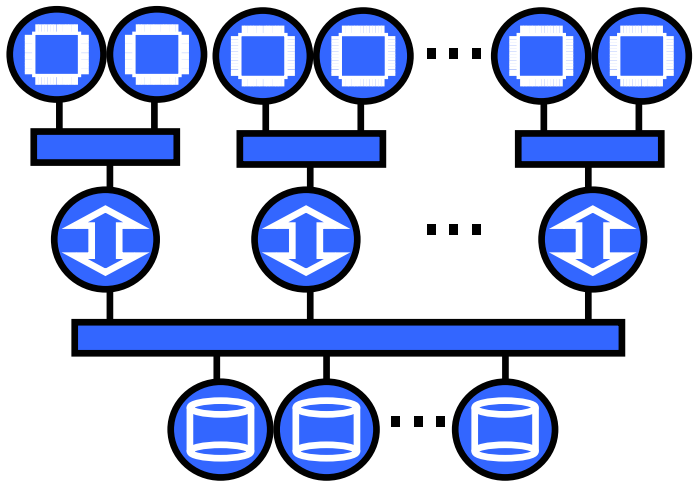


Software-based replication

- Inter-server communication may be leveraged to create replicas of objects
  - Servers with independent storage are paired
  - Clients talk to primary server
  - Modifications are forwarded to mirror
  - In case of failure, mirror can take over
  - Active/Active approach is applicable
- Compared to hardware-supported failover
  - Significantly less expensive
  - Higher latency due to network synchronization
- Work by Brad Settlemyer, Clemson Univ.



Very large scale systems



Processors running applications (10,000s-100,000s)

Support nodes (1000s)

I/O devices or servers (100s-1000s)

- These systems will have nodes between processors and storage
- Simply forwarding operations would be a disaster
  - 64K independent file create requests? Writes to the same file block?
- Aggregation, statelessness, lockless solutions, leveraging semantics, and failure tolerance will be key
  - Opportunity to apply all these concepts at both stages
- PVFS2, ROMIO MPI-IO are designed to meet the demands of this scale
  - Smaller systems should be easy in comparison!
- Infrastructure needed for support nodes – what should it look like?

Other interesting topics

- Wide-area file system access
  - Building on inter-server communication
  - New BMI implementation over UDP?
- Consistency semantics
  - What/how can we cache on clients and still be useful for computational science?
- Local storage organization
  - Should we manage blocks, use CAS, or use O_DIRECT?
- Security
  - How do we leverage existing security components in PVFS2?

PVFS2 is ready for use

- Runs on a wide variety of architectures, Linux kernels, and interconnects
- People have been beating on it for months now with a wide variety of tests
  - Averaging 80+ downloads per month (as many as PVFS1)
  - 85 PVFS2-users subscribers
- Performance will improve, but it isn't bad now (multi-GB/sec large I/O)
- Documentation, support mailing lists, etc. are all in place
- Failover, pvfs2-fsck, and GUI monitoring help complete the package

Additional information

- PVFS2 web site: http://www.pvfs.org/pvfs2
  - Documentation, mailing list archives, and downloads
- PVFS2 mailing lists (see web site)
  - Separate users and developers lists
  - Please use these for general questions and discussion!
- Internet Relay Chat (IRC)
  - Server irc.freenode.net, channel #pvfs2
  - Talk directly with developers
- Email
  - Rob Ross <rross@mcs.anl.gov>
  - Walt Ligon <walt@clemson.edu>
  - Pete Wyckoff <pw@osc.edu>
  - Phil Carns <pcarns@parl.clemson.edu>
  - Neill Miller <neillm@mcs.anl.gov>
  - Rob Latham <robl@mcs.anl.gov>

PVFS2

PARALLEL VIRTUAL FILE SYSTEM

Thanks for coming!

Any Questions?

http://www.pvfs.org/pvfs2