

Future directions in large-scale storage systems

Justin M Wozniak

Argonne National Laboratory

Presented at:

Northwestern University

Evanston, IL – May 3, 2010

Outline

- Overview
 - High-performance computing and systems software
 - Exascale on the horizon
- Next-generation filesystems
 - Object storage systems
 - Distributed data structures (C-MPI)
- Reliability at extreme scale
 - Data placement for survivability
 - Simulation and analysis of rebuild performance (GOBS)
- Many-file applications
 - Swift and many-task computing
 - Improvements for data access to many small files (CDM)

High-performance computing

- Leadership systems
 - ANL - IBM BG/P *Intrepid* @ 557 TFlops
 - ORNL - Cray XT5 *Jaguar* @ 1.75 PFlops
 - TACC - Sun Constellation *Ranger* @ 505 TFlops
- Clusters
 - U of Chicago - Intel Xeon *PADS* - 48 nodes x 4 cores
 - ANL - AMD *Breadboard* - 64 nodes x 8 cores
- Grids
 - Open Science Grid - ~25,000 nodes
 - TeraGrid - Access to a variety of high-performance resources

Uses of high-performance storage (1)

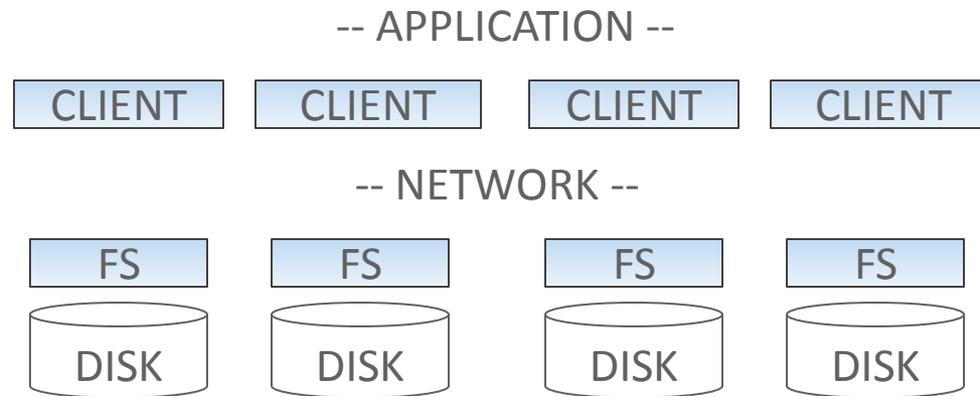
- Checkpoint
 - Write out all user memory to non-volatile storage
 - Basic survival strategy to avoid lost work
 - Optimal checkpoint interval
 - First-order approximation to optimal checkpoint write interval
- $$t_o = \sqrt{2t_w t_f}$$
- » t_o : checkpoint interval
 - » t_w : time to write checkpoint
 - » t_f : mean time to failure
- Future trends
 - Bigger memory → longer writes
 - More components → more faults
 - Could reach a critical point

Uses of high-performance storage (2)

- Useful application data
 - MPI-IO
 - Parallel interface for file I/O operations
 - Allows I/O experts to implement optimizations
 - High-level libraries
 - Provide a variable-oriented view on data
 - PnetCDF, HDF5, ADIOS
 - Can use MPI-IO
 - POSIX I/O
 - Still prevalent in large-scale applications
 - Must maintain user expectations, portability, but make use of high-performance machines

Parallel filesystems

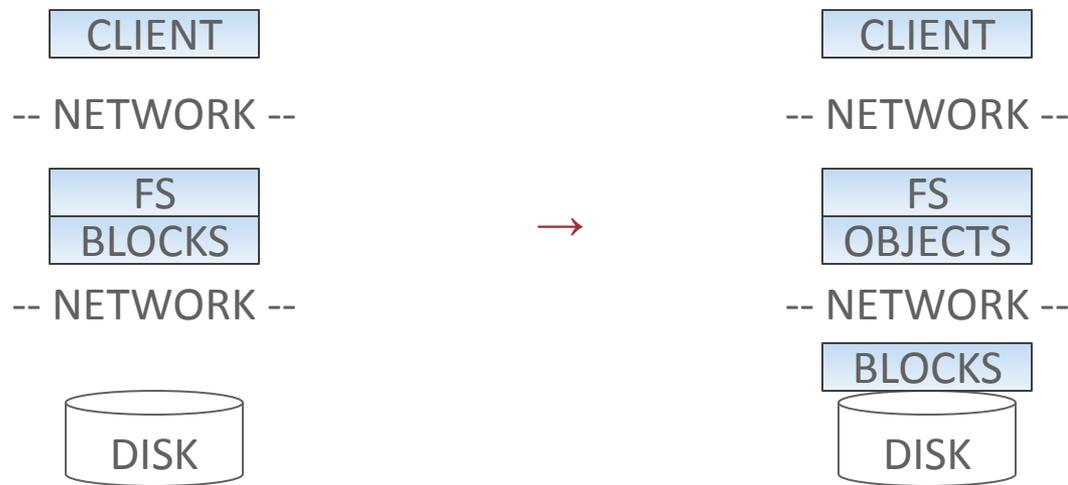
- Eliminate single bottlenecks in I/O



- PVFS – Clemson, ANL
 - Open source, community maintained
- GPFS – IBM
 - Licensed by IBM
- Lustre – Oracle/Sun
 - Open source but supported
- PanFS – Panasas
 - Software/hardware packages

Object storage

- Separation of concerns

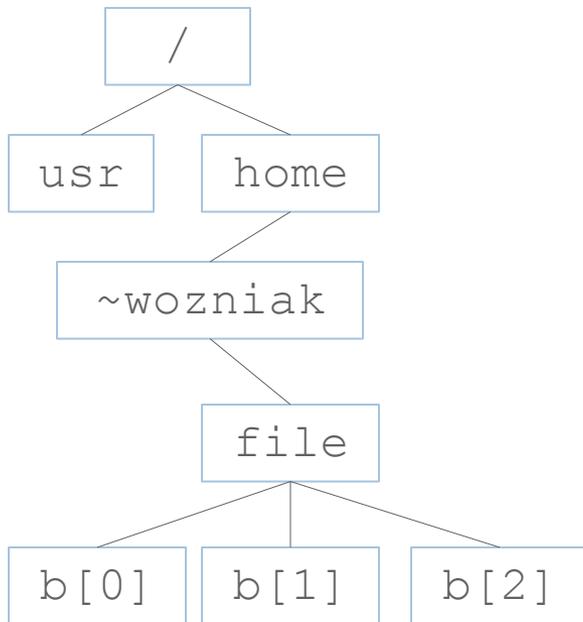


- Employed by many modern systems – not “old news” either

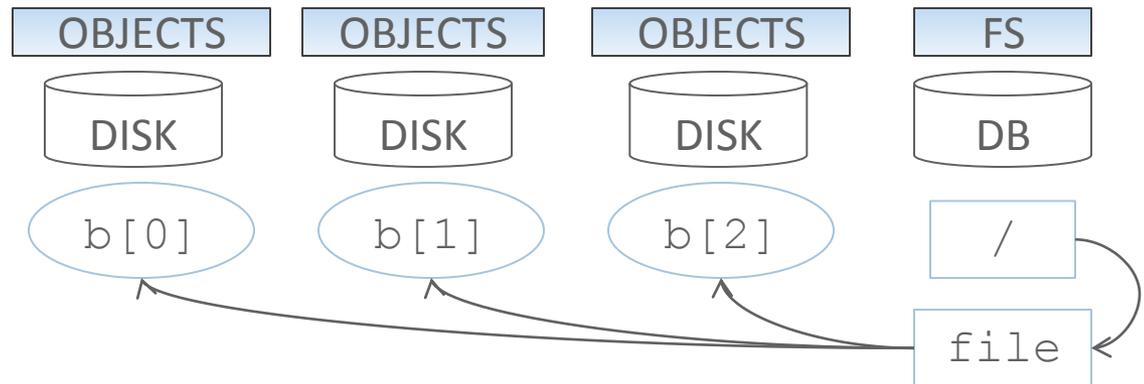
Distributed data structures



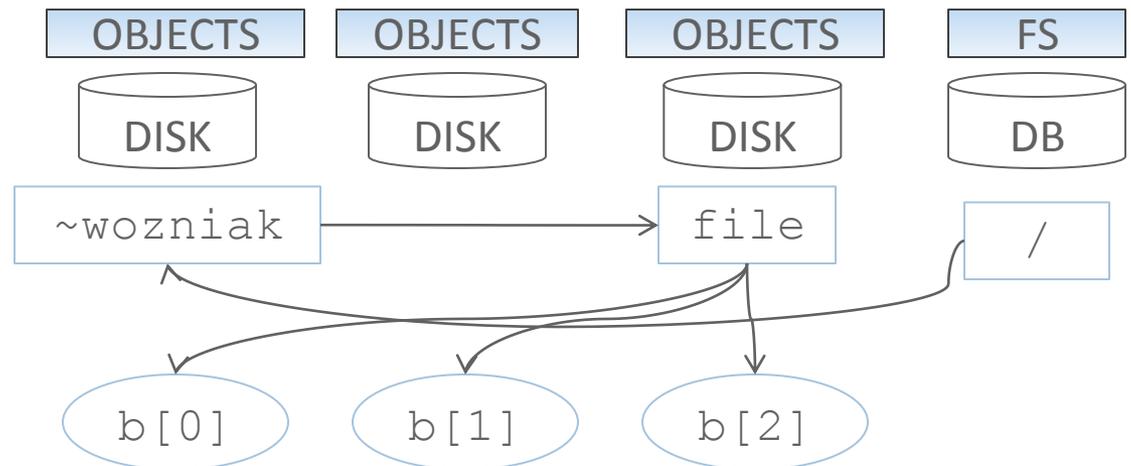
Data placement in parallel filesystems



- Centralized metadata



- Distributed metadata



Distributed metadata

- Design features
 - Decentralized (fast)
 - Reliable
 - Distributed
 - Consistent
- Distributed hash tables (DHTs)
 - Originally designed for wide-area networks
 - Self-constructing
 - Self-organizing
 - Self-healing
 - Scalable
 - ...
 - Chord
 - Pastry
 - Kademlia
 - CAN
 - ...

Server-server communication

- Metadata
 - Distributed objects (distributed directories)
 - Searches

- Collective operations
 - Allocation of striped files
 - Control communication

- Network choices
 - BMI
 - Network abstraction layer
 - Developed for PVFS, now a stand-alone system
 - MPI
 - Rich API for parallel programming
 - Typically used by high-performance applications

Content-MPI (C-MPI)

- New DHT implementation based on MPI
 - MPI library allows integration with existing software, methods
 - Abstraction over DHT details, placement algorithm

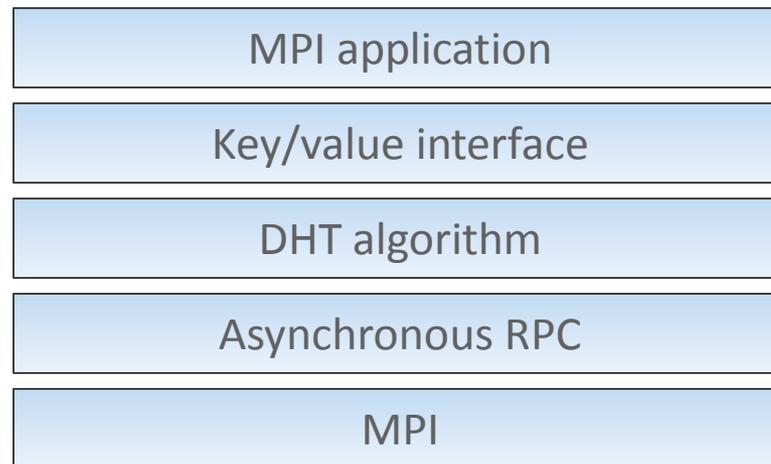
- Uses “monolithic” MPI or “dynamic processes” without application modification
 - Monolithic
 - “Normal” MPI usage
 - Uses one big MPI communicator
 - Dynamic processes
 - MPI-2 feature set
 - Dynamic processes allow for dynamic allocation and connection of independent processes
 - Basic model for fault-tolerant MPI programs

C-MPI use cases

- MPI library
 - Programming model analogous to Linda, blackboards
 - Perform remote function on remote object
 - Maintain critical application state in distributed, fault tolerant manner
- Distributed database
 - Perform lookups for key/value pairs
 - Check on state of application progress
- Shell IPC
 - Shell tools provided to communicate with background process linked to C-MPI
 - Useful for many-task computing (more to come...)

C-MPI internals

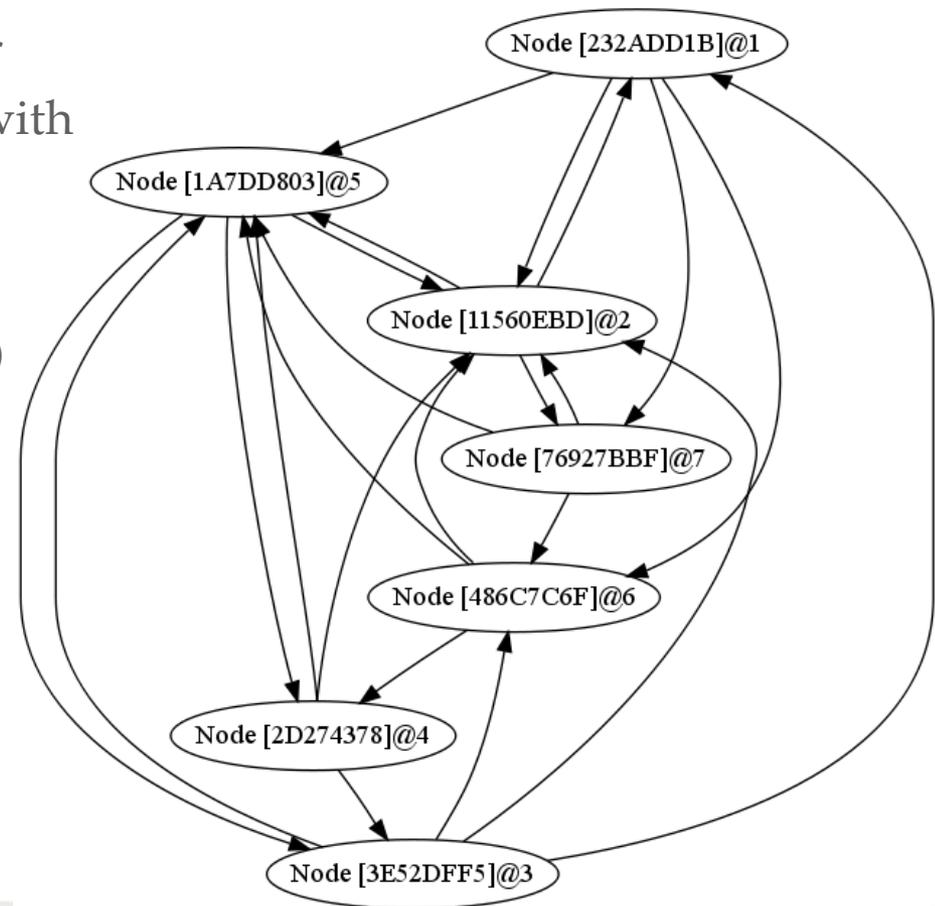
- Layered architecture



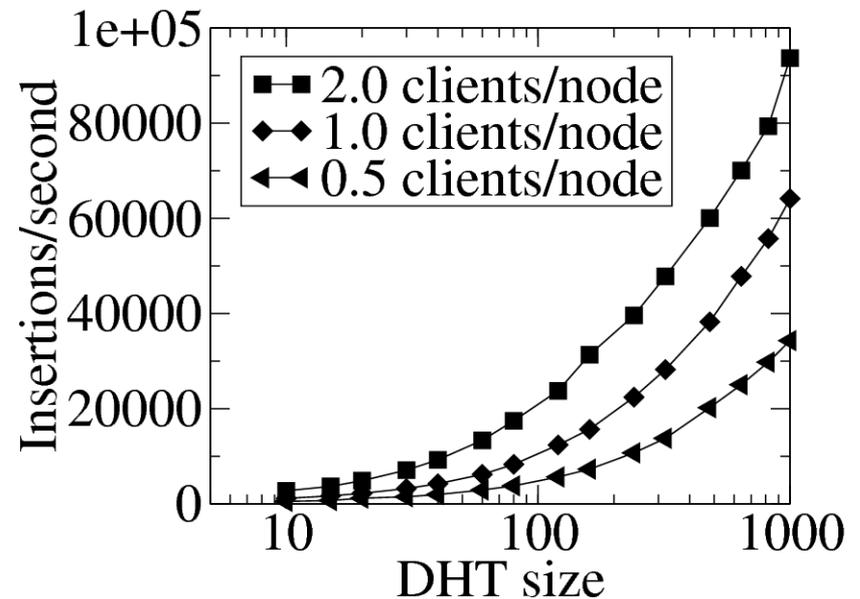
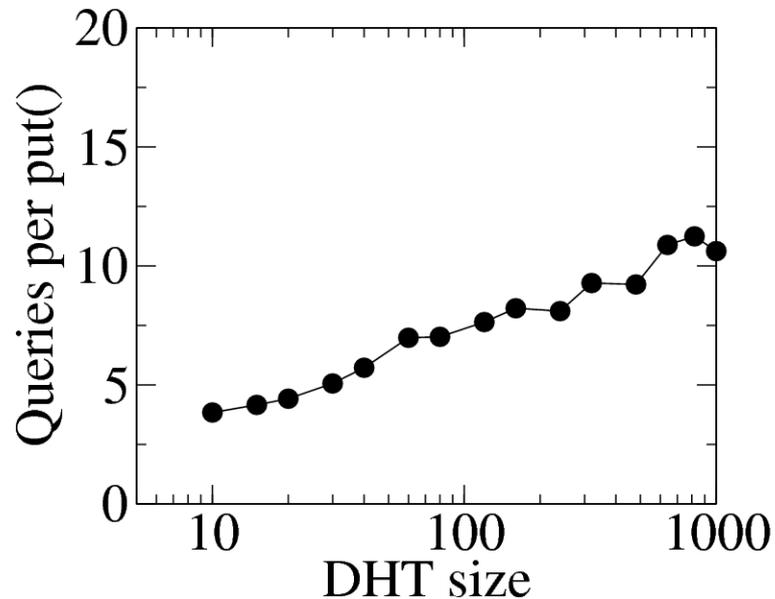
- MPI-RPC programming model
 - Use non-blocking MPI calls
 - Make progress on RPC return using user function pointer
 - Help with management of many outstanding RPCs

Kademlia

- Assign each node a 160-bit identifier
- Use XOR metric:
distance(X,Y) = xor(X,Y) as integer
- Each node stores a neighbor table with $O(\log n)$ rows, k columns:
 - For node X,
Row i contains k nodes Y :
distance(X,Y) = xor(X,Y) $\in [2^i, 2^{i+1})$
- New neighbors discovered dynamically



Performance results: SiCortex

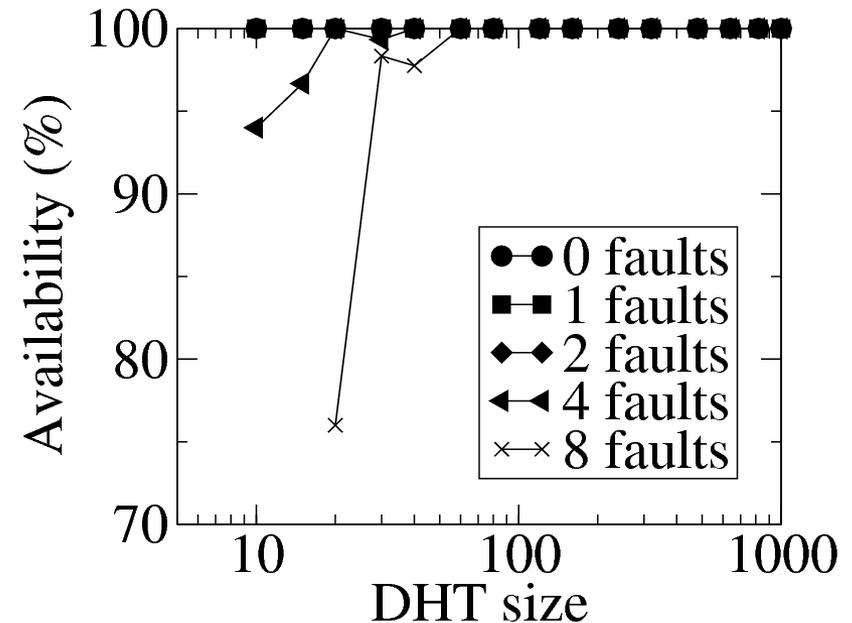


- Lookup RPCs per user lookup

- Small key/value pair insertions; memory only

Fault-tolerance

- Fault emulation in MPI-RPC
 - User sets node to emulate failure
 - Subsequent RPCs “fail”
- Performs as expected
 - Data still available
 - Overlay network not partitioned



- Probability that a key/value pair is still available given a system size and fault count
- Replica count = 3

Fault-tolerance in MPI

- MPI Standard
 - Overall assumption: MPI users should not worry about faults
 - Standard does allow communication errors to be reported to user
 - Theoretically could recover from errors on one communicator, continue to use and create other communicators: this is our approach
 - Difficult issues remain in the case of collective operations, blocking operations (cannot wait forever)
 - May be addressed by new non-blocking collective operations and dynamic process functionality
- MPI implementations
 - Typically, cannot recover from errors
 - Work is being done...

C-MPI: Summary

- Distributed storage requires highly scalable metadata management
- Distributed hash tables
- Implementation

Object storage rebuild simulation

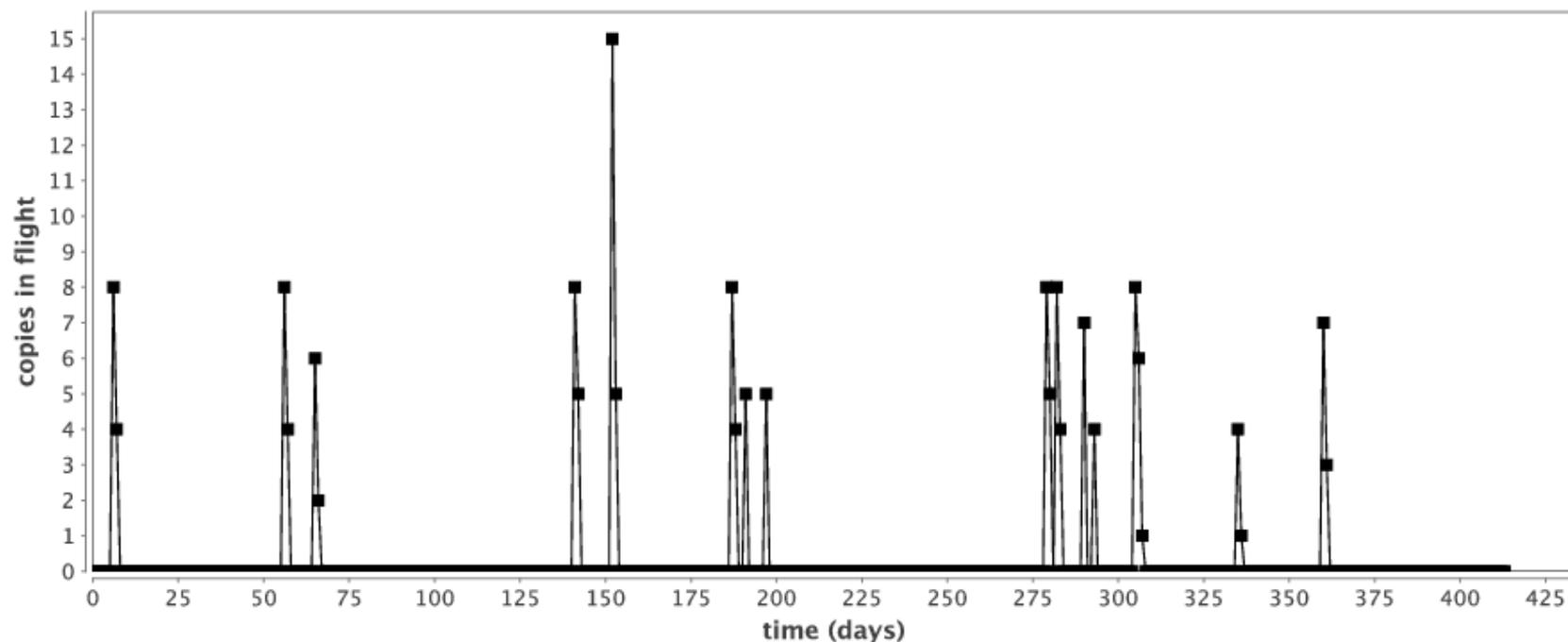


Exascale storage challenges

- Number of disks
 - Speed: to satisfy checkpoint requirements, will need ~30,000 disks
 - Capacity: may use additional storage hierarchy for space
- Required bandwidth
 - ~12 TB/s
 - New ability to manage many clients
- Redundancy
 - Must plan to lose up to 10% of disks per year
 - That's 263 TB/day; 3.125 GB/s
- (Power)

Disk failure rates

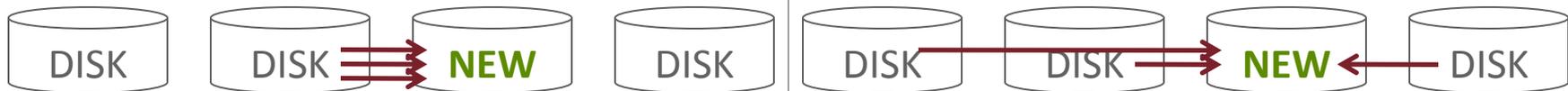
- CMU study
 - Typically ~5%/year
 - Up to 13%
- Google study
 - Below 5% in first year
 - Peaks near 10% in year 3



- GOBS simulation of 32,000 disks in RAID 5 (4+1)
Plot shows inter-node traffic due to RAID loss

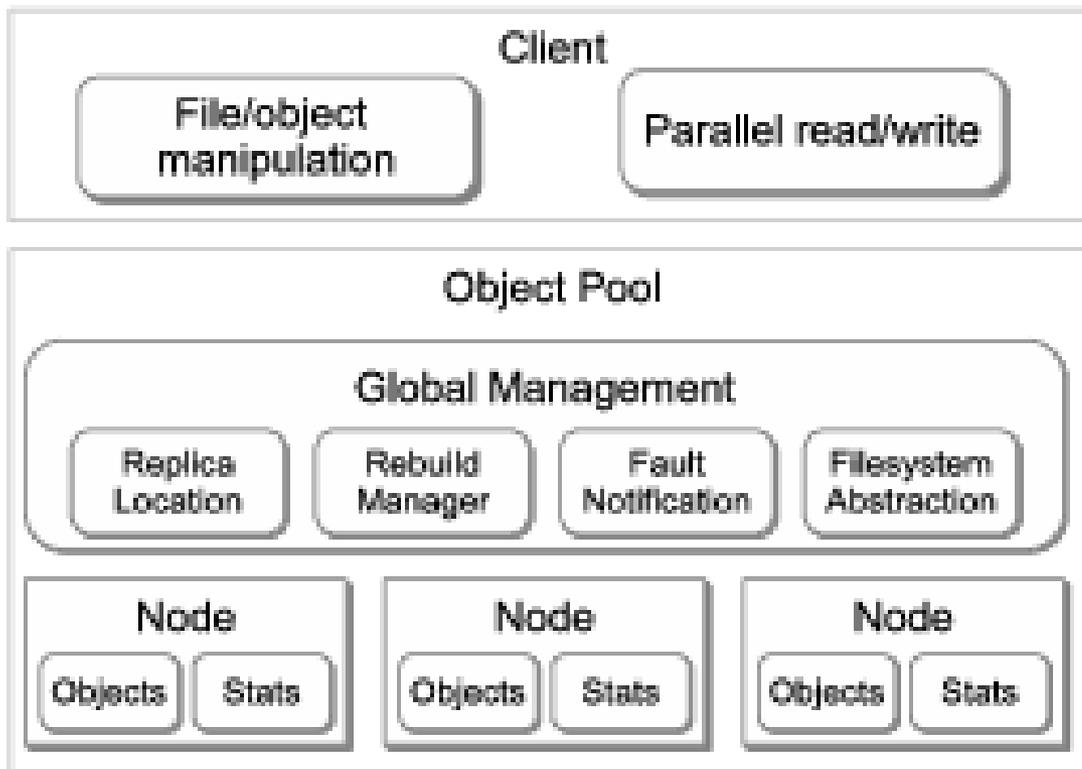
Simple data placement is problematic

- Combine local RAID with inter-node replication for availability
- Local RAID is relatively faster for read-modify-write operations
- Whole node loss – often temporary – managed with replicas
- Replica chaining
 - Simple, localized object placement
 - On rebuild, creates a hot spot of activity
- Large declustered RAIDs
 - Fully distributed
 - On rebuild, all nodes involved, all write to one new disk



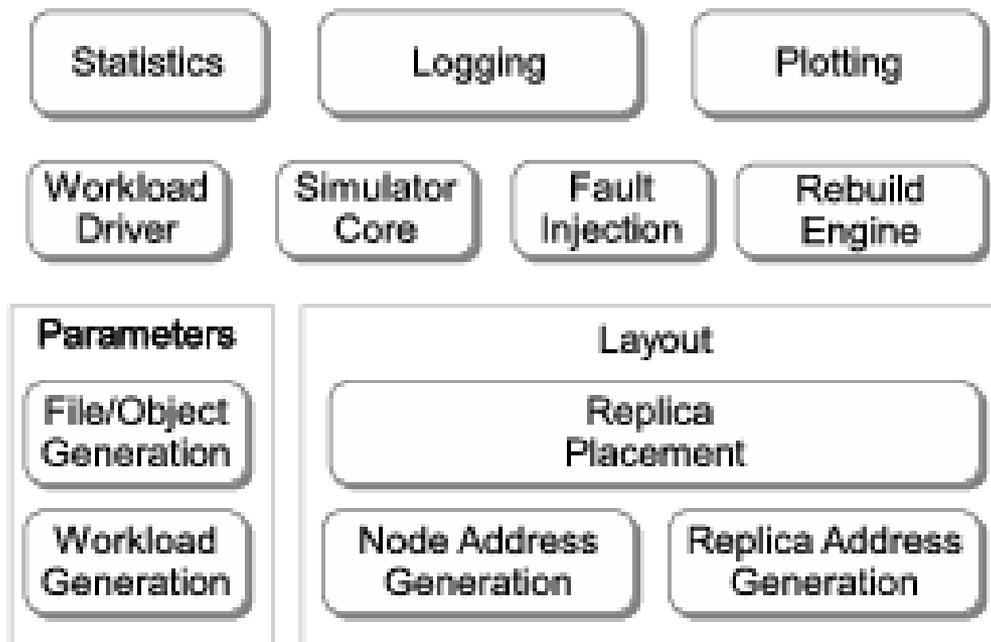
Simulation as initial approach

- Simulated system



- Workload simulation
- Idealized control
- Object servers

- General Object Space (GOBS) simulator architecture

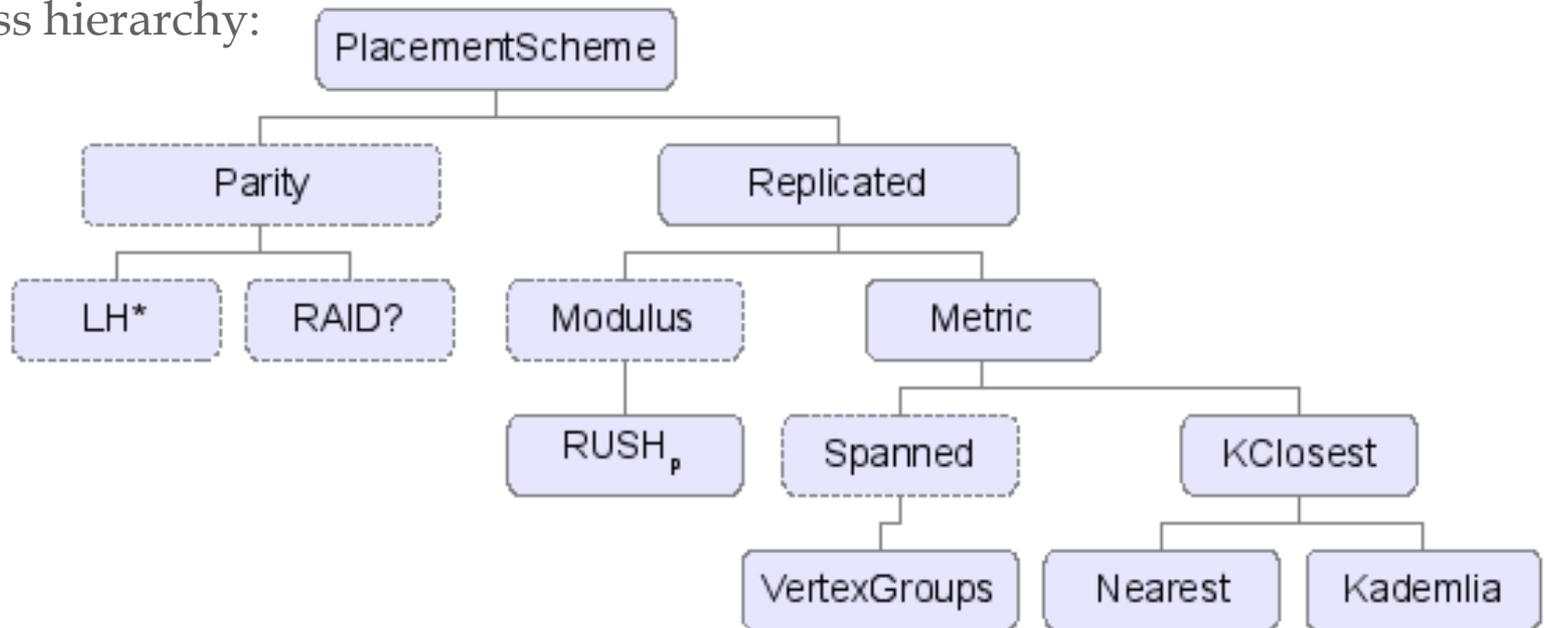


- User interface
- Core functionality
- Replaceable components

Simulator - extensibility

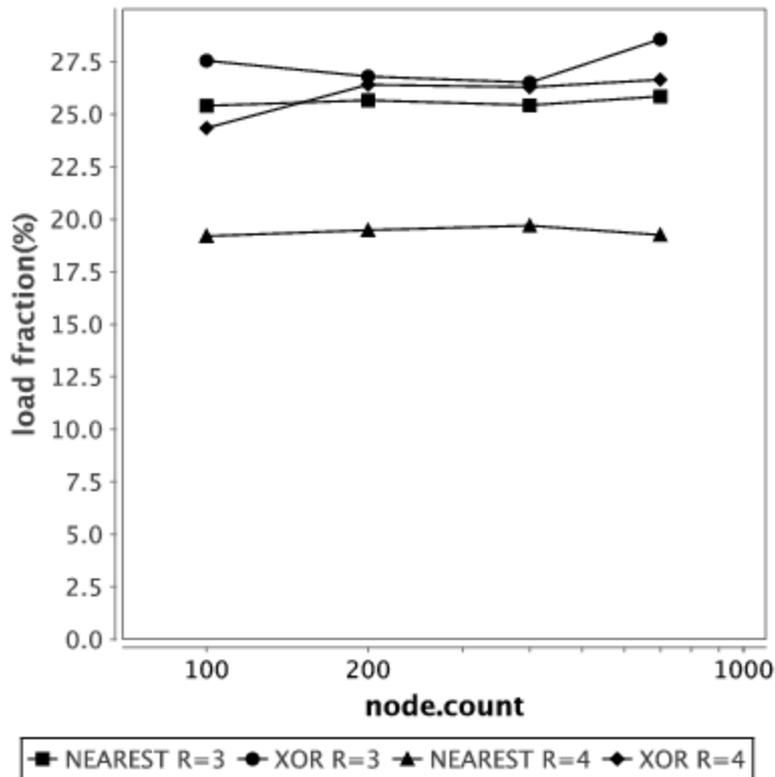
- Extensible Java simulator
 - Heavy use of inheritance
 - Enable easy implementation of new schemes

- Class hierarchy:

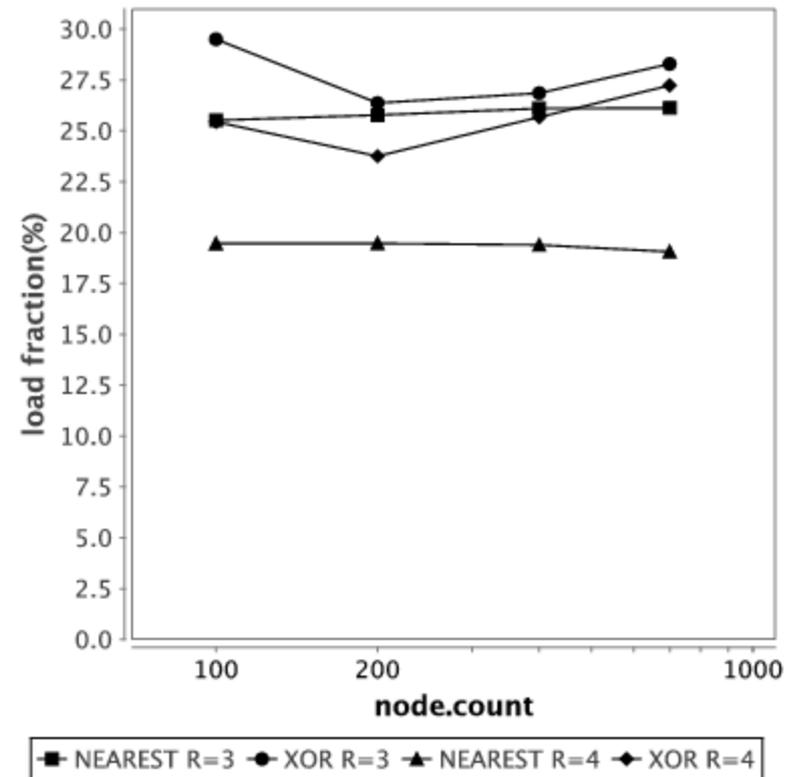


GOBS results - rebuild hot spots

- 600 servers; 30 TB disks; RAID 5 (4+1); disk transfer rate 400 MB/s;
- 1EB filesystem
- Single fault induced - rebuild performed



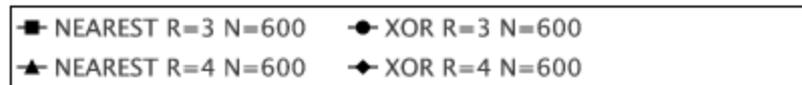
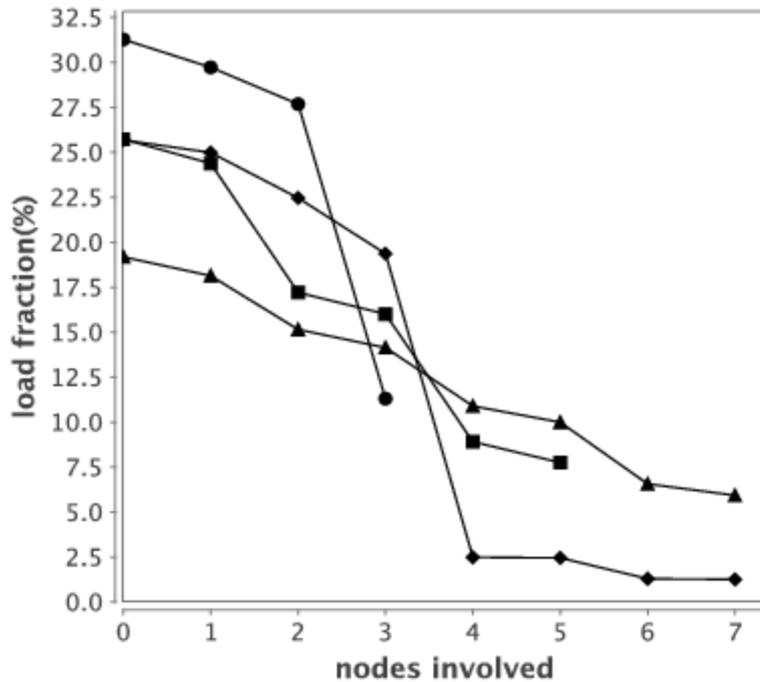
- Replica pulled from last in chain



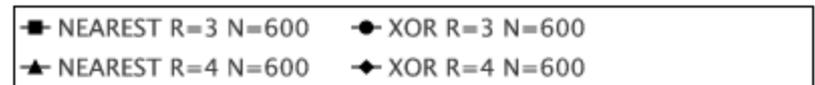
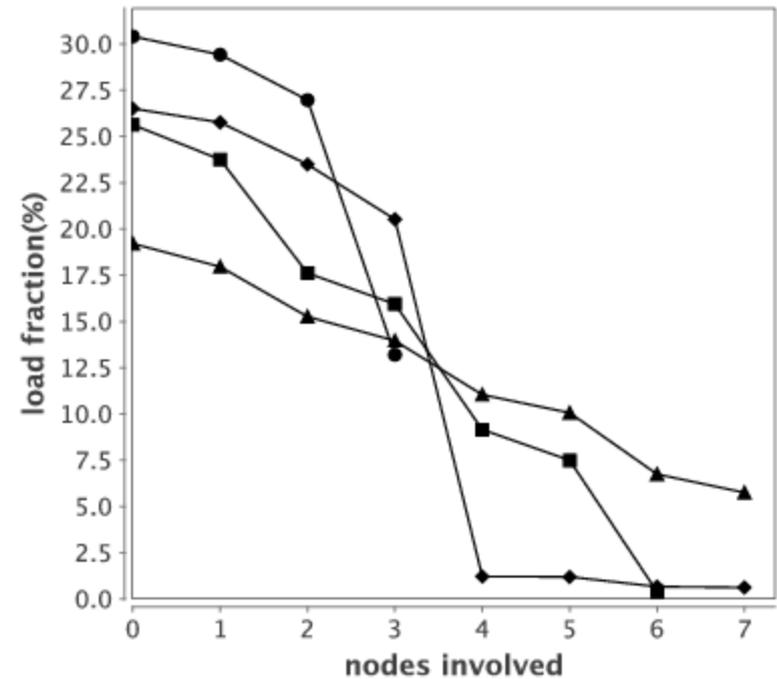
- Replica pulled from random node

GOBS results – rebuild curves

- Single fault induced – rebuild performed



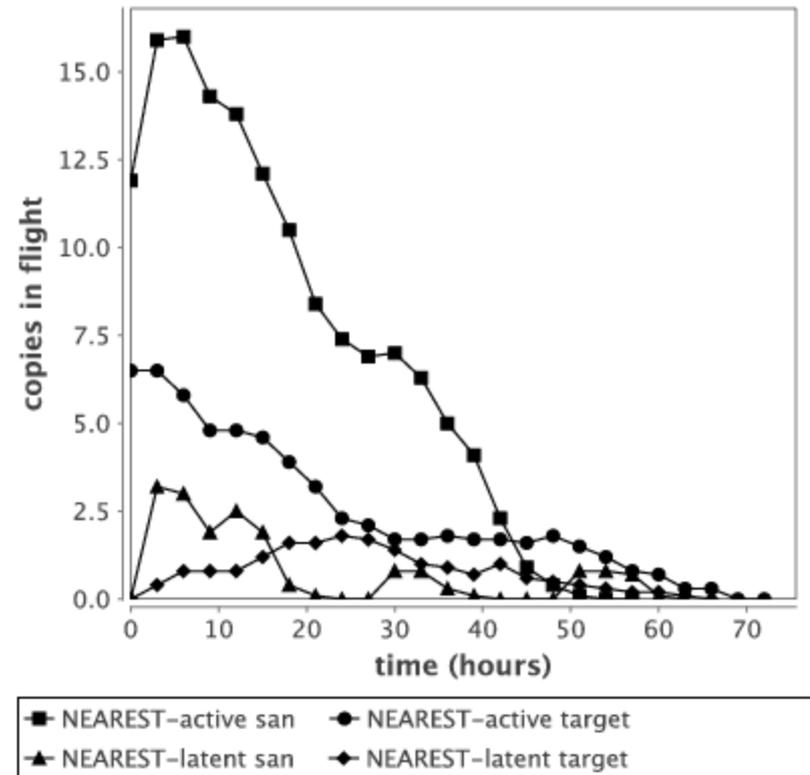
- Replica pulled from last in chain



- Replica pulled from random node

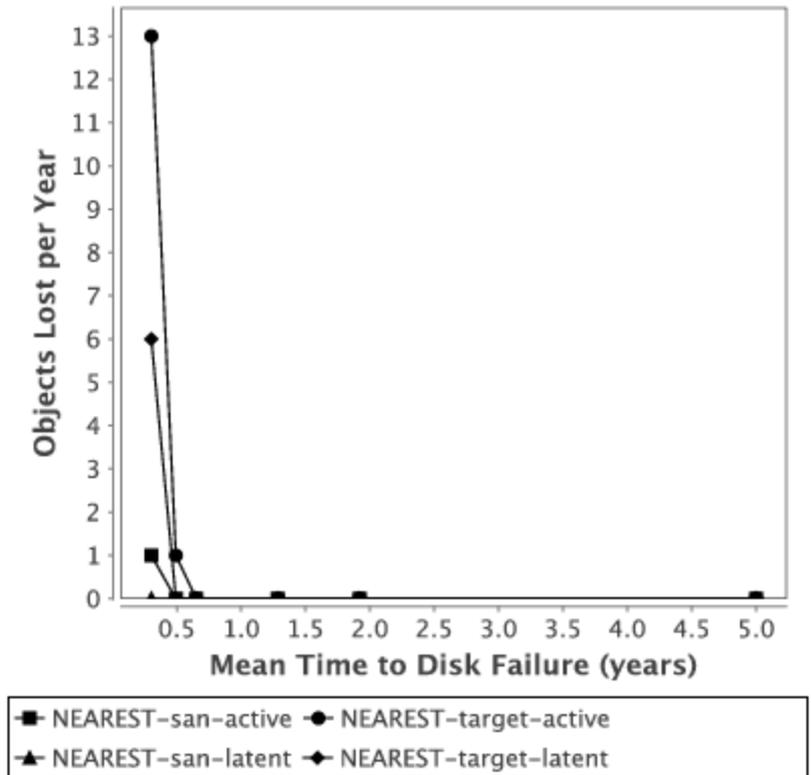
GOBS results – rebuild concurrency

- Multiple faults induced – average traffic recorded
- Replica pulled from primary
- “target” – RAID (4+1)
- “san” – RAID (8+2)
- “active” – begin copies immediately
- “latent” – wait until replacement is inserted



GOBS results – data loss

- Vary disk MTTF and report objects lost per year
- Neither scheme loses data unless MTTFs are extremely low
- Indicates that aggressive schemes may be used that favor user accesses
- (How does one quantify amount of data loss?)



GOBS: Summary

- Data placement strategies matter when performing rebuilds
- Rebuild time matters over long data lifetimes
- Simulation can help evaluate placement strategies
- Much more to do here...

Collective data management



Many-task computing

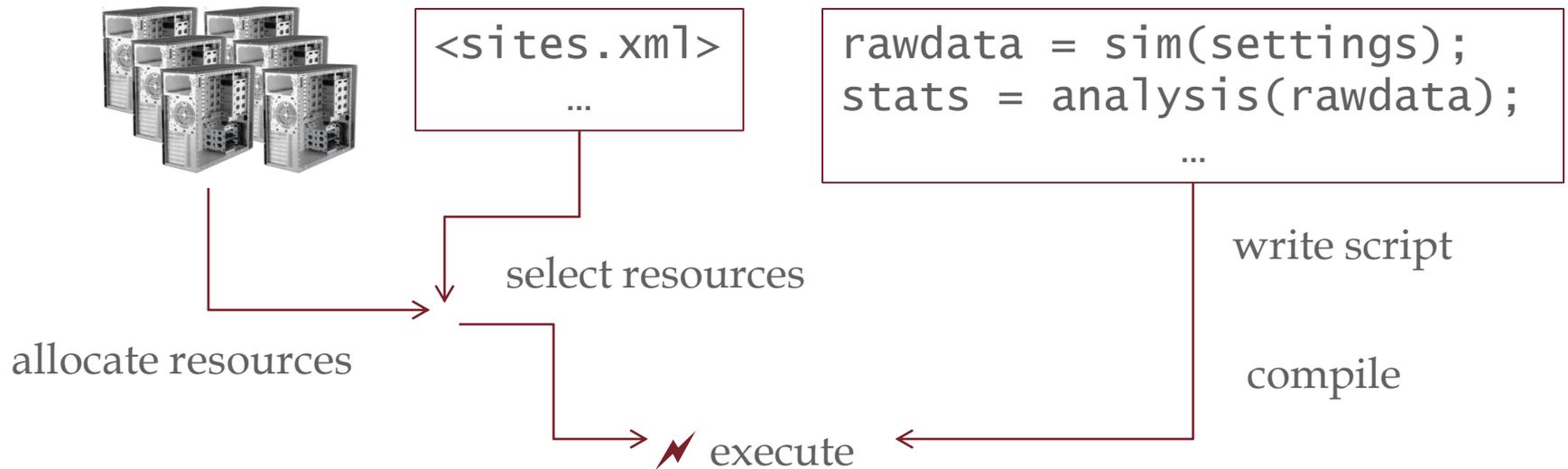
- Combine grid computing infrastructure with high-performance resources
- Reuse robust grid software systems
- Use new, rapid schedulers (Falcon, Coasters)
- Plenty of applications
- Fault-tolerant, scalable

Scripted applications

- Development timeline:
 - Scientific software developer produces sequential code for application research
 - Produces small batch runs for parameter sweeps, plots
 - Small scale batches organized through the shell and filesystem
 - Additional scaling possible through the application of grid tools and resources
 - 🖐️ What if the application is capable of (and worthy of) scaling further?

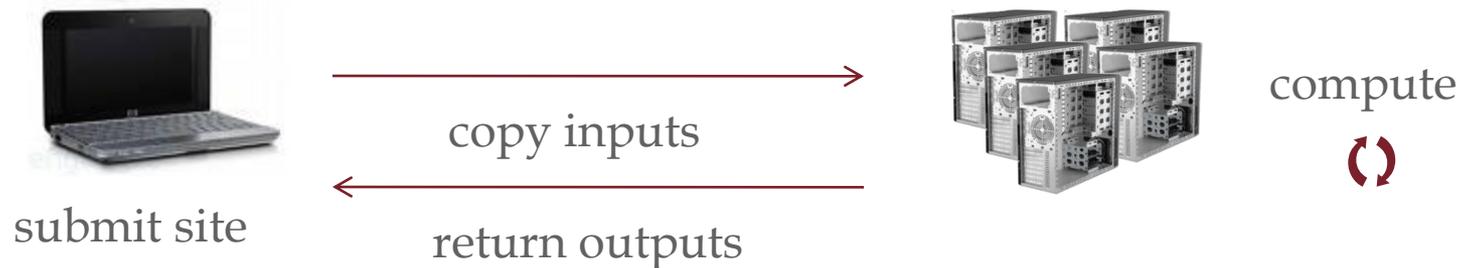
Swift and related tools

- Separate workflow description from implementation
- Compile and generate workloads for existing execution infrastructures



Default I/O

- In a standard Swift workflow, each task must enumerate its input and output files
- These files are shipped to and from the compute site



- This RPC-like technique is problematic for large numbers of short jobs

Data generation and access

- Current I/O systems work recognizes the challenges posed by large batches of small tasks
- Characterized by:
 - Small files
 - Small, uncoordinated accesses
 - Potentially large directories
 - Whole file operations
 - Metadata operations
 - File creates
 - Links
 - Deletes
- Overall challenges
 - BlueGene/P:
 - I/O bandwidth: down to 400 KB/s /core
 - File creation rate: only 1/hour /core (Raicu et al.)

Related work

- Filesystem optimizations
 - PVFS optimizations for small files (Carns et al. 2009)
 - Improved small object management
 - Eager messages
 - BlueFS client optimizations (Nightingale et al. 2006)
 - Speculative execution in the filesystem client
 - Mitigates latency
- Scheduling and caching
 - BAD-FS (Bent et al. 2004)
 - Data diffusion (Raicu et al. 2009)
- Collective models
 - Enable programmer support
 - Borrow from strengths of MPI, MPI-IO functionality
 - Expose patterns explicitly (MapReduce, etc.)

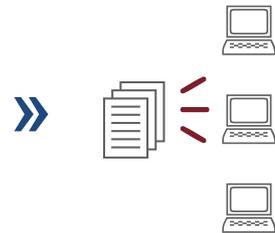
Collective Data Management

- Provide primitives that the programmer can use explicitly
 - May already be used via custom scripts
 - Generally difficult to specify with sequential languages

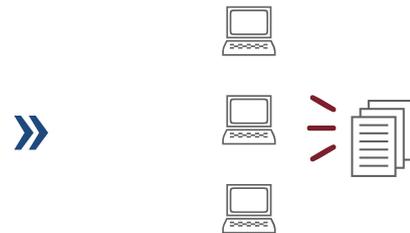
- Broadcast (aggregation, map):



- Scatter (two-phase):



- Gather (aggregation, reduce)

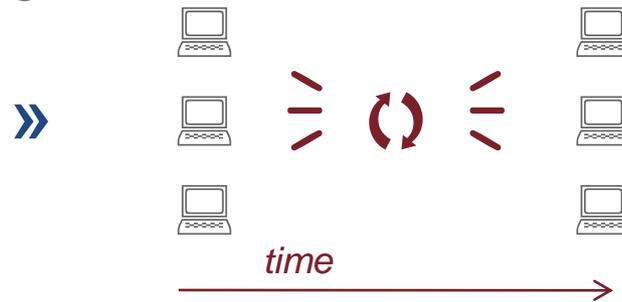


Cache techniques

- Cache pinning (specify critical data)

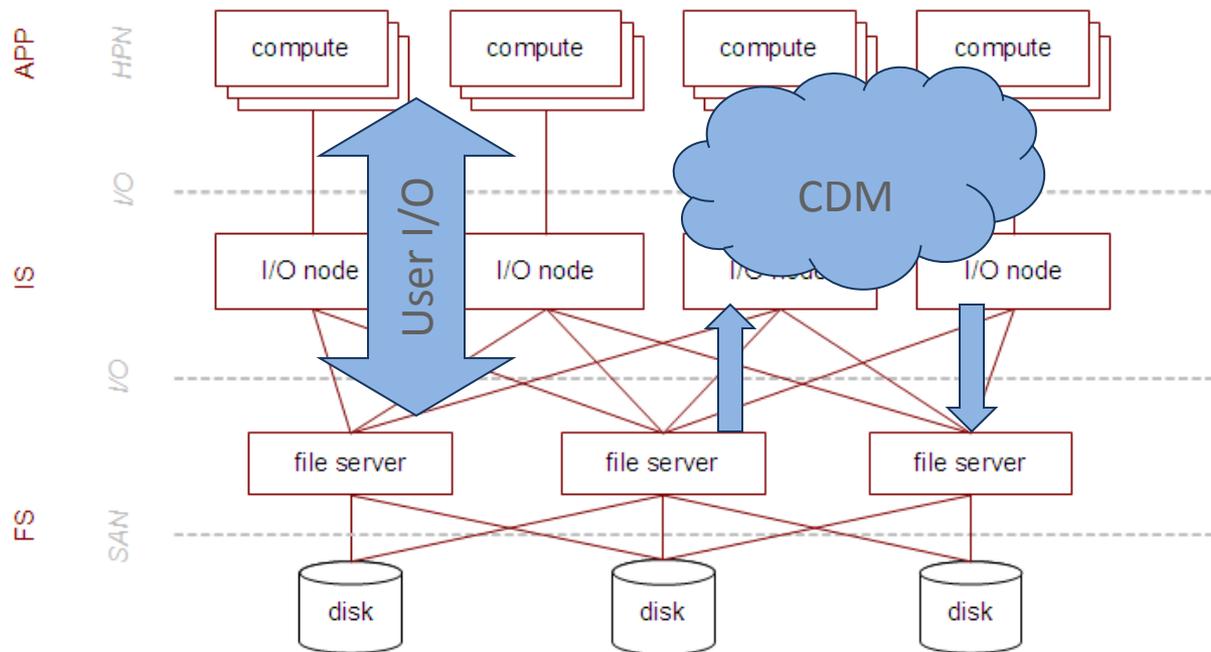


- Workflow / data-aware scheduling



I/O reduction

- Let applications continue to move large quantities of small data over POSIX interfaces
- Prevent these accesses from reaching the filesystem



I/O reduction

- The purpose of each potential CDM technique is to reduce accesses to the filesystem
- In our case studies, we sought to estimate the maximum possible reduction that a carefully-written application could achieve on our target system model
- In a default scripted workflow, all accesses go to the FS
- As a start, we used an I/O reduction defined as:

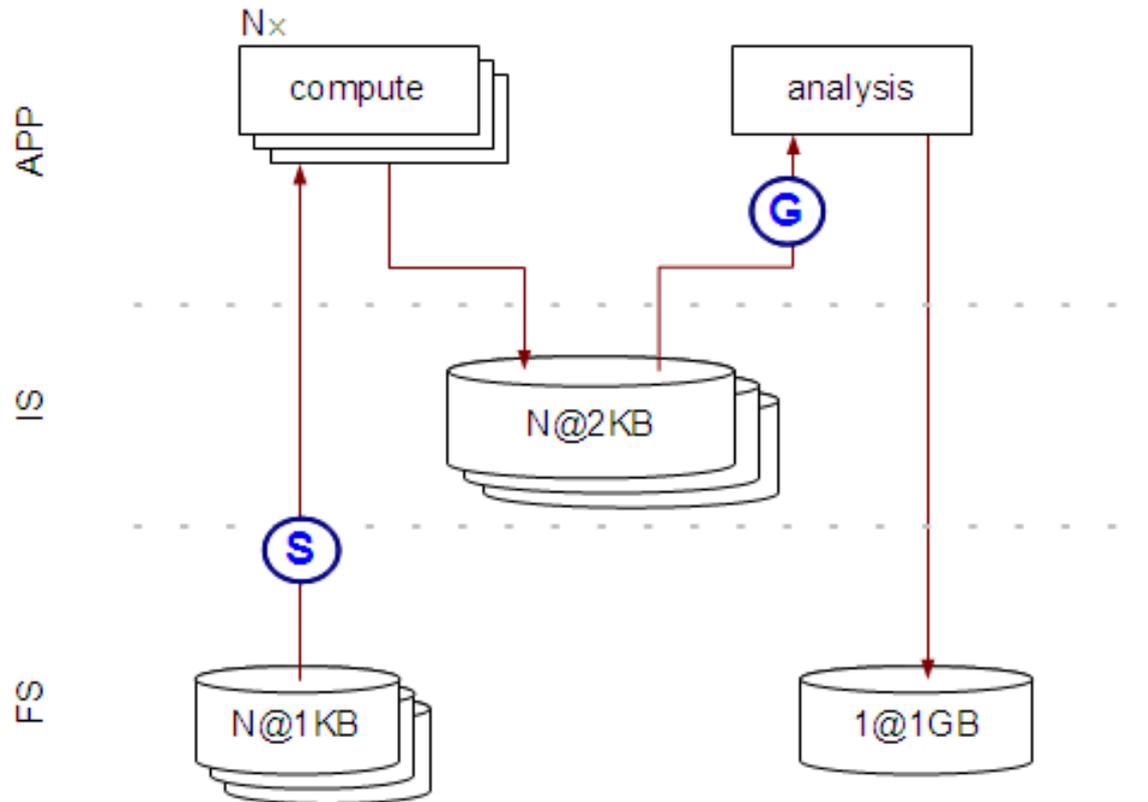
$$\text{reduction} = 100\% - \frac{\text{I/O seen by FS}}{\text{I/O seen by apps}} \quad \text{▪ in bytes}$$

- Other interesting quantities could measure file creates, links, or a count of accesses regardless of size

Case studies: High-level view

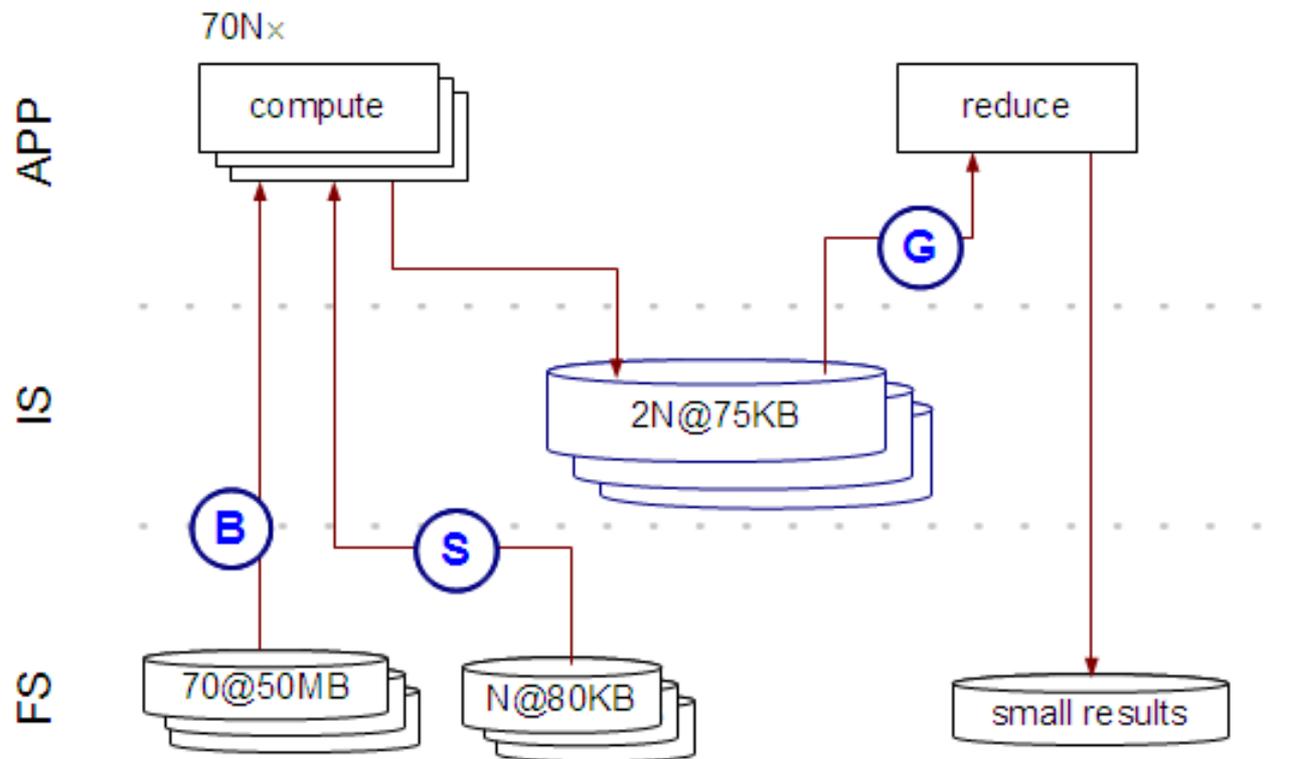
- OOPS: Open Protein Simulator
- DOCK: Molecular docking
- BLAST: Basic Local Alignment Search Tool
- PTMap: Post-transformational modification analysis
- fMRI: Brain imaging analysis

fMRI



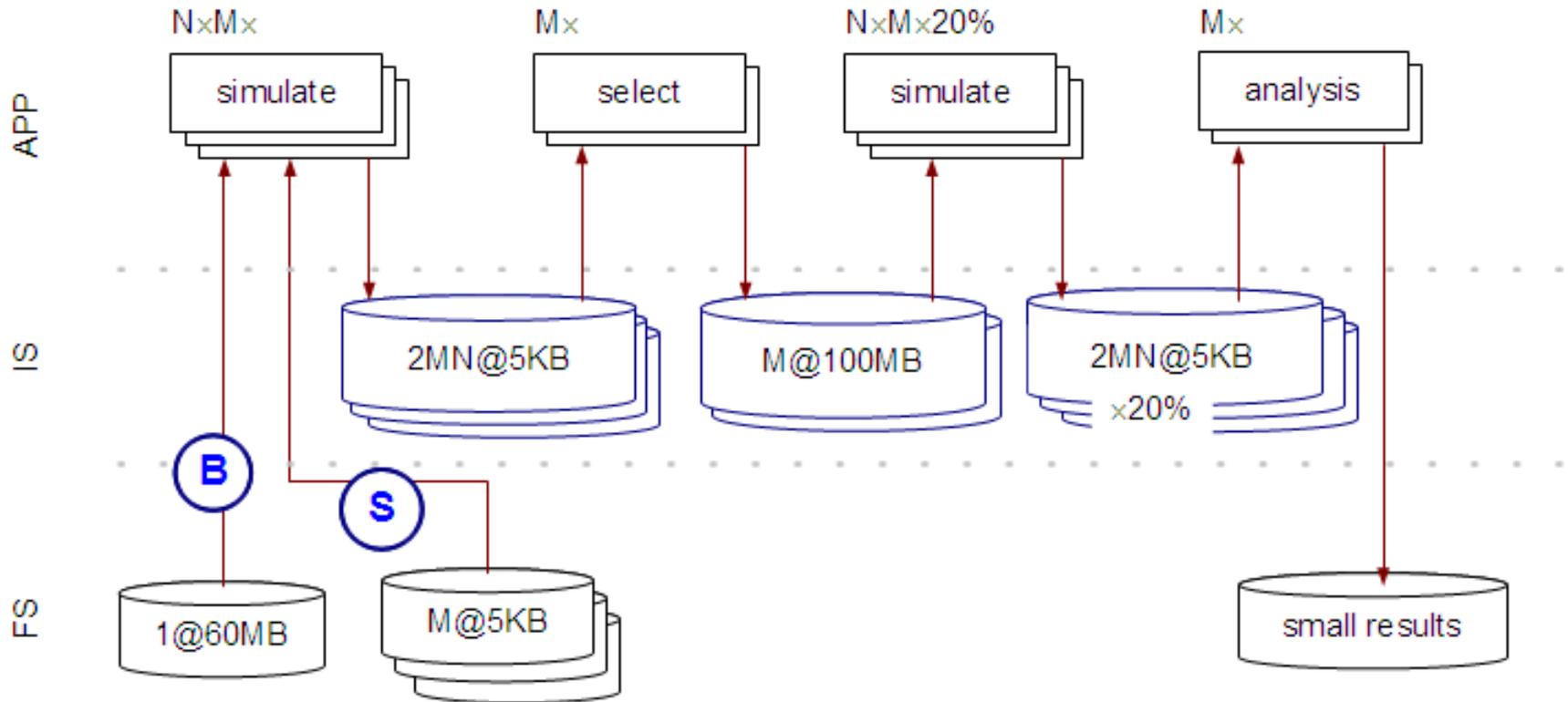
- Simple MapReduce-like structure
- Broken down into **scatter** and **gather** operations
- Intermediate data can be cached. Produces much final output

BLAST



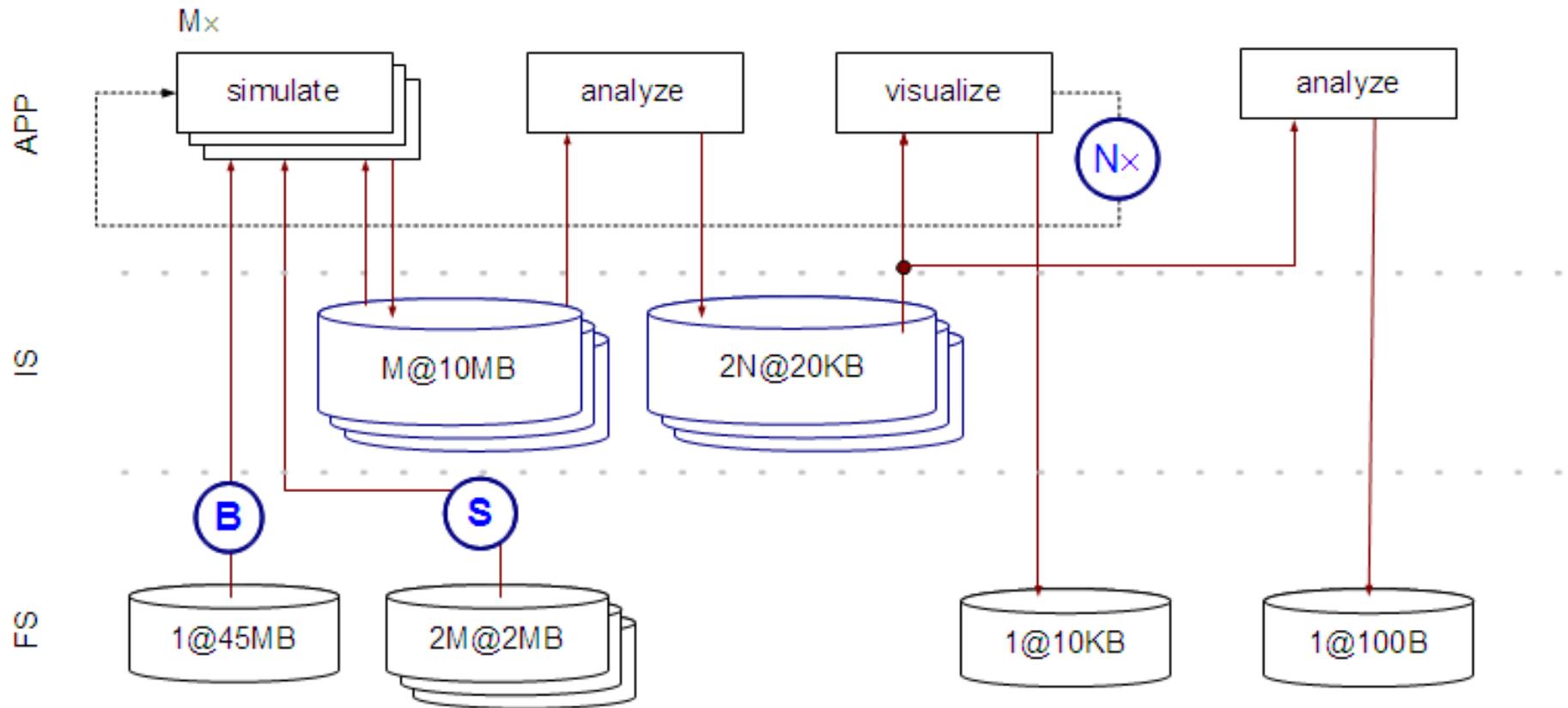
- Like MapReduce with two inputs
- If cache is used to implement **broadcast**, must prevent pollution
- Produces trivial final output - I/O reduction may exceed 99%

DOCK



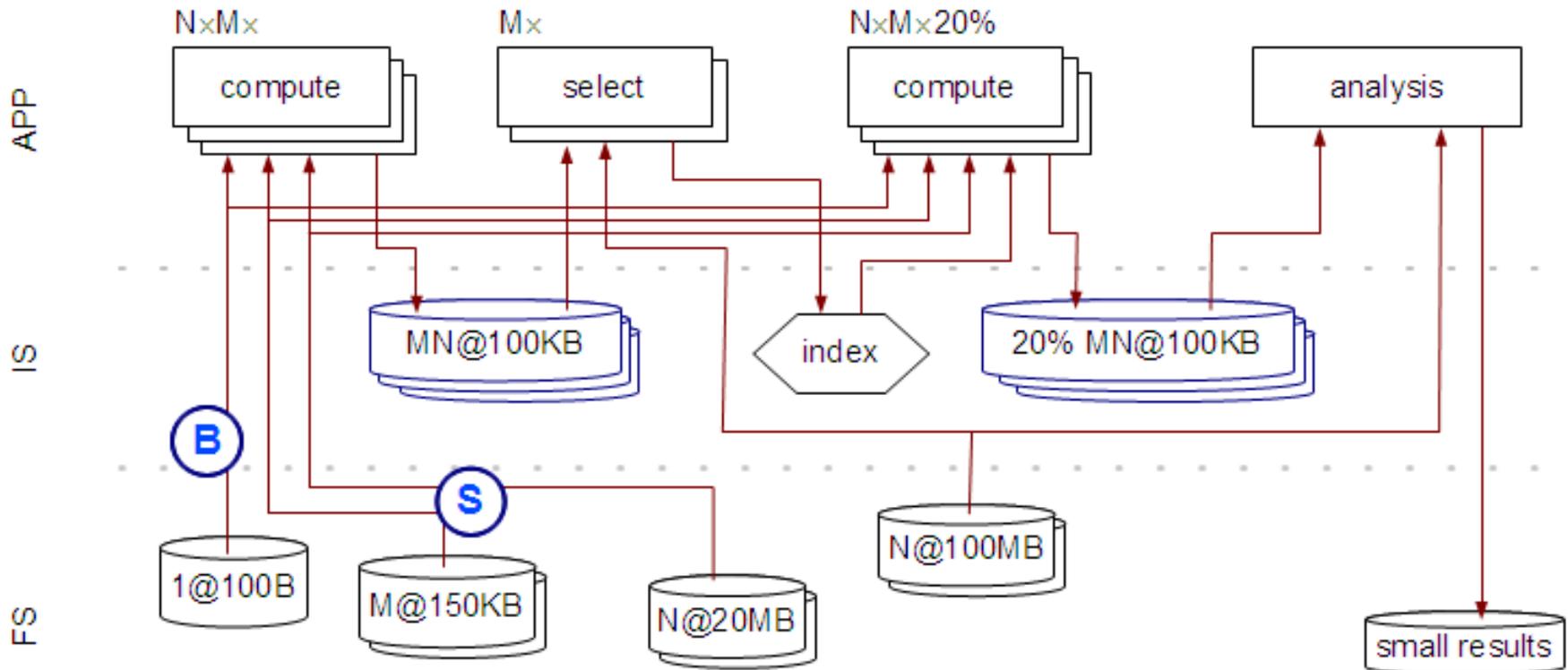
- Significant input size
- Pipeline-like accesses
- Produces trivial final output - I/O reduction may exceed 99%

OOPS



- Significant input size
- Pipeline-like accesses and iterations
- Produces trivial final output - I/O reduction may exceed 99%

PTMap



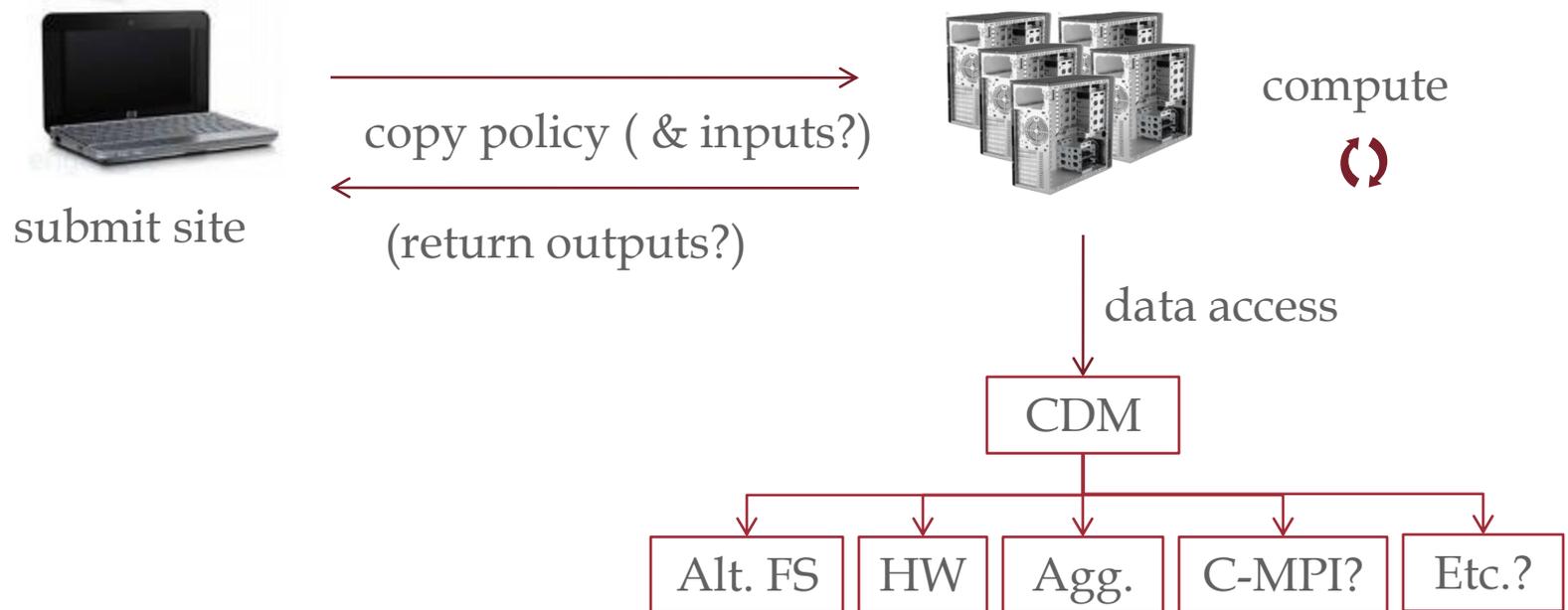
- Pipeline-like accesses and iterations
- Uses links to create an intermediate index
- Produces trivial final output - I/O reduction may exceed 99%

Observations

- Great deal of potential optimizations
 - Many of which are previously studied
 - Difficult to implement with sequential programming models
- Small files
 - Large input data sets must be read efficiently
 - Many small files are created, written once, and possibly read again multiple times, primarily by transmission to other compute jobs
 - Developer basically knows this – must be able to express it
- Patterns
 - MPI-like concepts such as broadcasts, gathers, and even point-to-point messages help describe the I/O patterns
 - Can be exposed to the developer through scripting abstractions

CDM active client

- New CDM module allows for dynamic data access on the compute site
- Implemented by modifying Swift wrapper scripts



Summary

- Investigated I/O performance characteristics of five scalable applications
 - Laid out workflow job/ data dependencies
 - Compared with well-studied patterns
 - Performed coarse studies of file access statistics
 - Looked at idealized potential optimizations (gedankenexperiments)
- Portability
 - Running on the BG/P not unlike running on the grid
 - Benefit from existing software systems
 - Work within the typical scientific development cycle
- Lots to do
 - Proposed new software toolkit and language integration
 - Largely based on existing tools; package and expose to developers

Thanks

- Rob Ross and the Radix group
- Mike Wilde and the Swift community
- Application collaborators: Yue Chen (PTMap), Aashish Adhikari (OOPS) and Sarah Kenny (fMRI)
- Thanks to Ioan Raicu and CUCIS
- Grants:

This research is supported in part by NSF grant OCI-721939, NIH grants DC08638 and DA024304-02, the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy under Contracts DE-AC02-06CH11357 and DE-AC02-06CH11357. Work is also supported by DOE with agreement number DE-FC02-06ER25777.

Questions