Case Studies in Storage Access by Loosely Coupled Petascale Applications

Justin M Wozniak and Michael Wilde

Petascale Data Storage Workshop at SC'09

Portland, Oregon – November 15, 2009

Outline

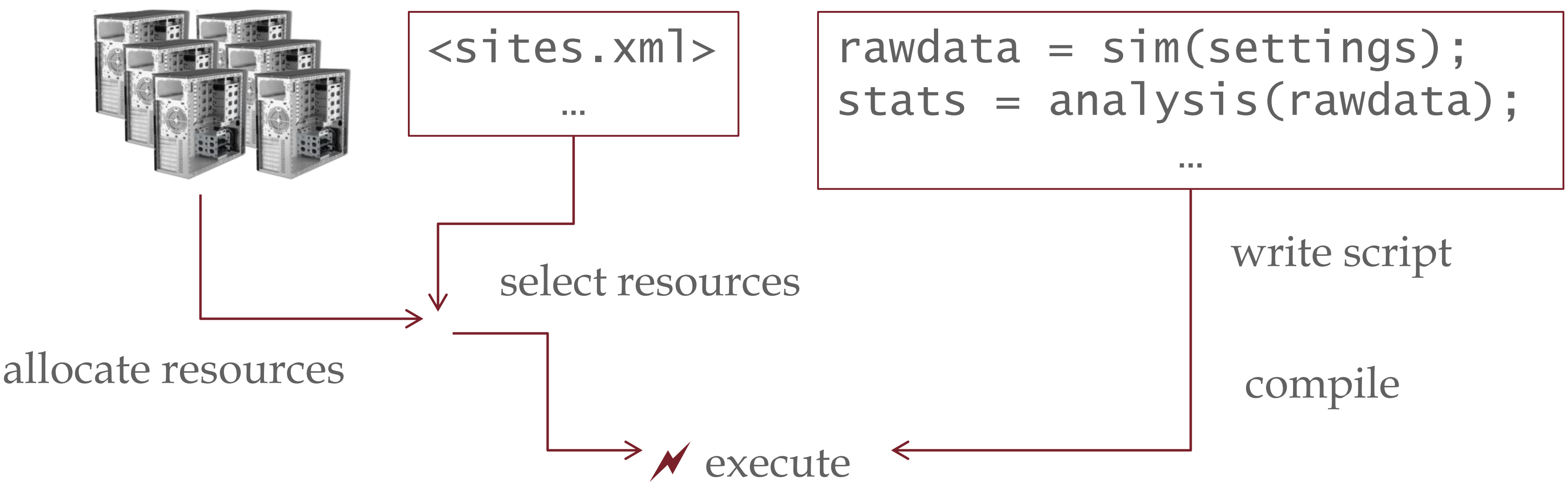
- Scripted scientific applications
  - Overview of parallel scripting
  - I/O challenges
  - Existing solutions and related work
- Collective data management
  - Communication and I/O model
  - Basis and theoretical benefit
- Case studies
  - High-level features
  - Look for well-studied patterns
- Summary

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
  - This model scales up to about an 8 node cluster
  - Additional scaling possible through the application of grid tools and resources
  - ☞ What if the application is capable of (and worthy of) scaling further?

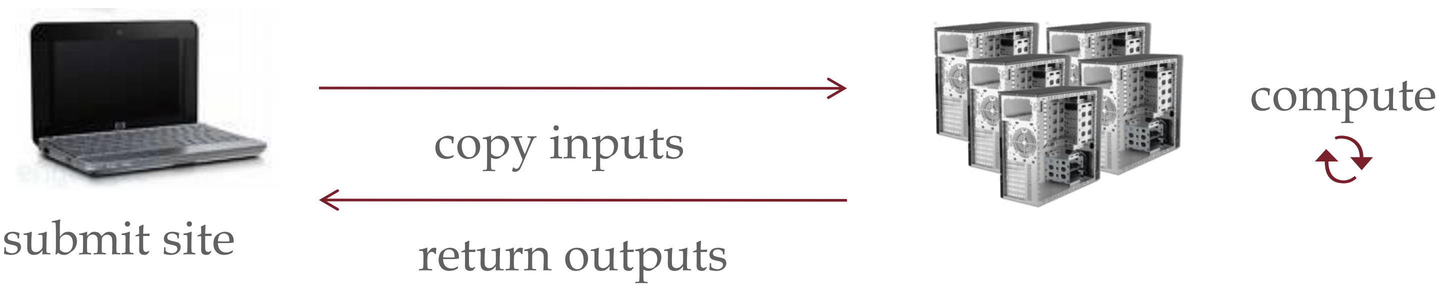
Swift and related tools

- 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.)

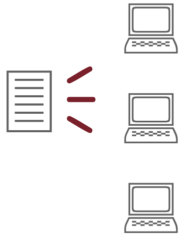
Related work

- 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.)

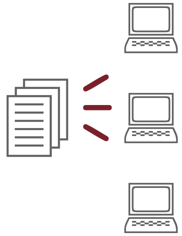
Collective Data Management

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

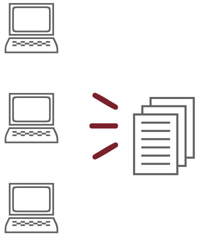
- Broadcast (aggregation, map):



- Scatter (two-phase):



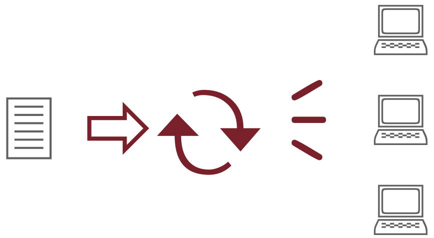
- Gather (aggregation, reduce)



Cache techniques

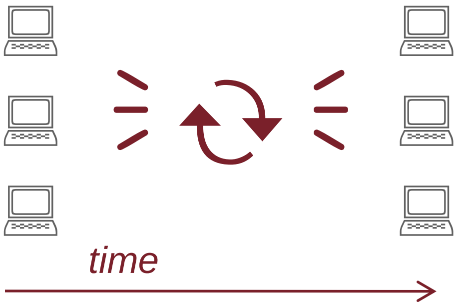
- Cache pinning (specify critical data)

»



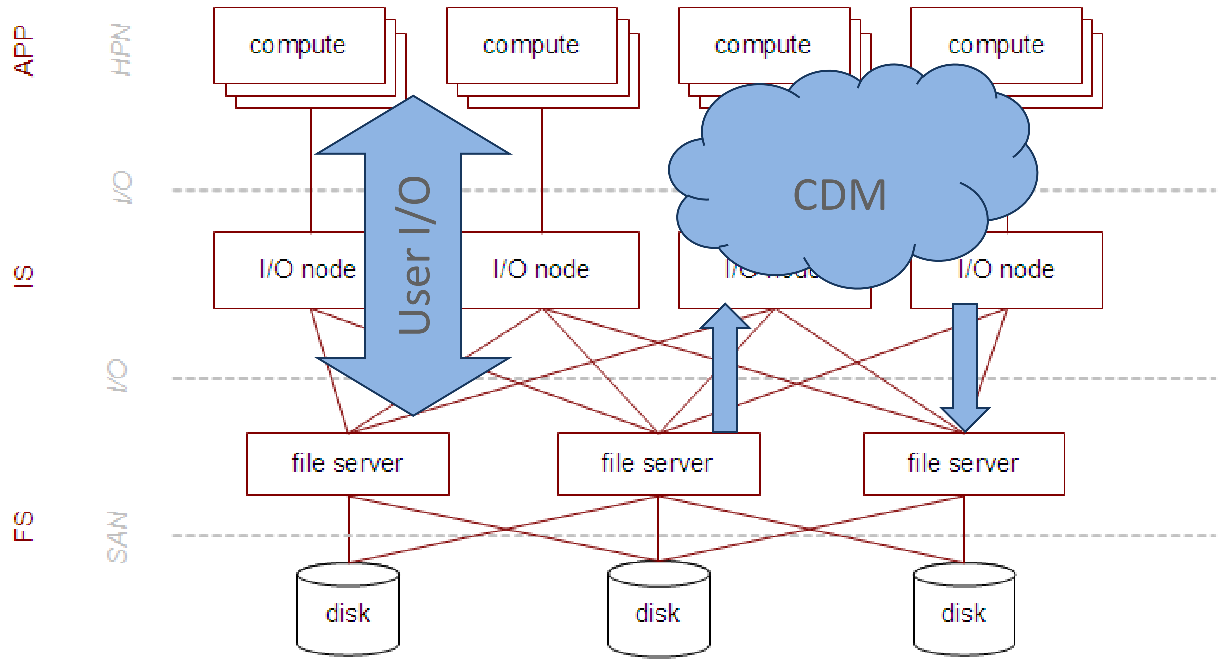
- Workflow/data-aware scheduling

»



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:

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

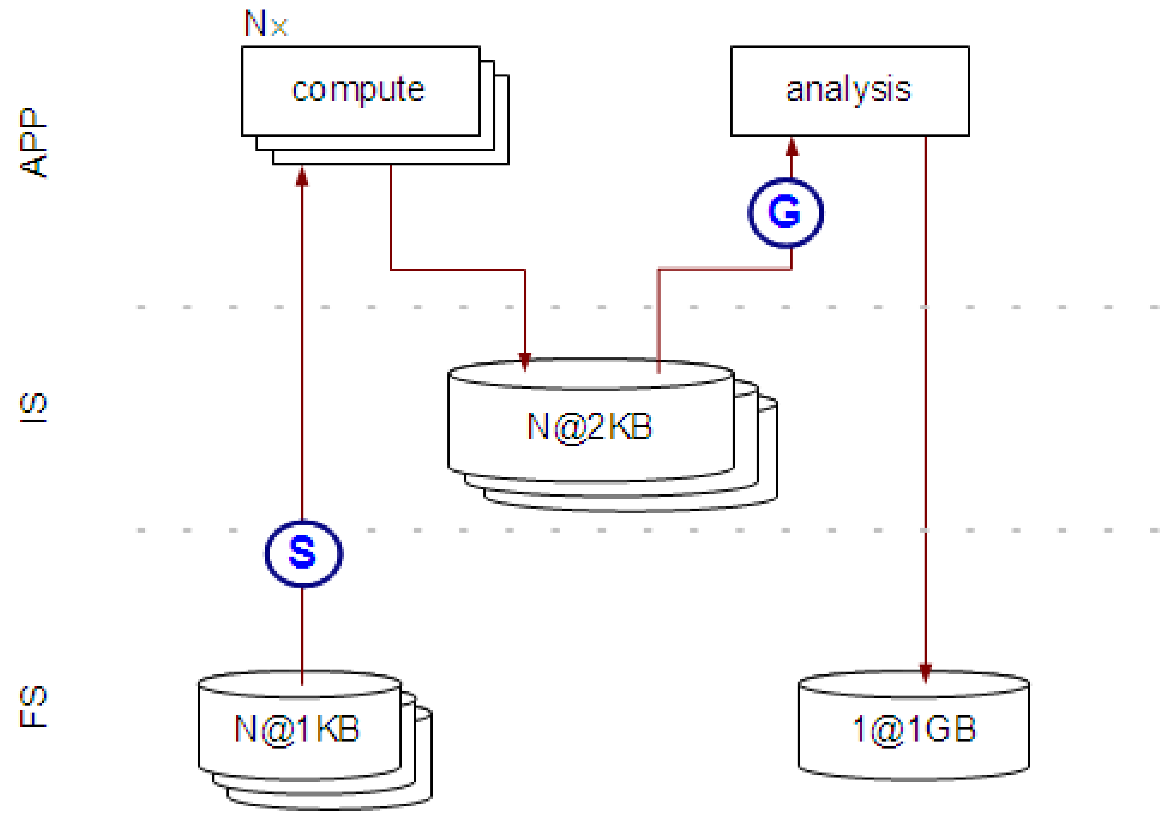
- in bytes

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

Case studies: High-level view

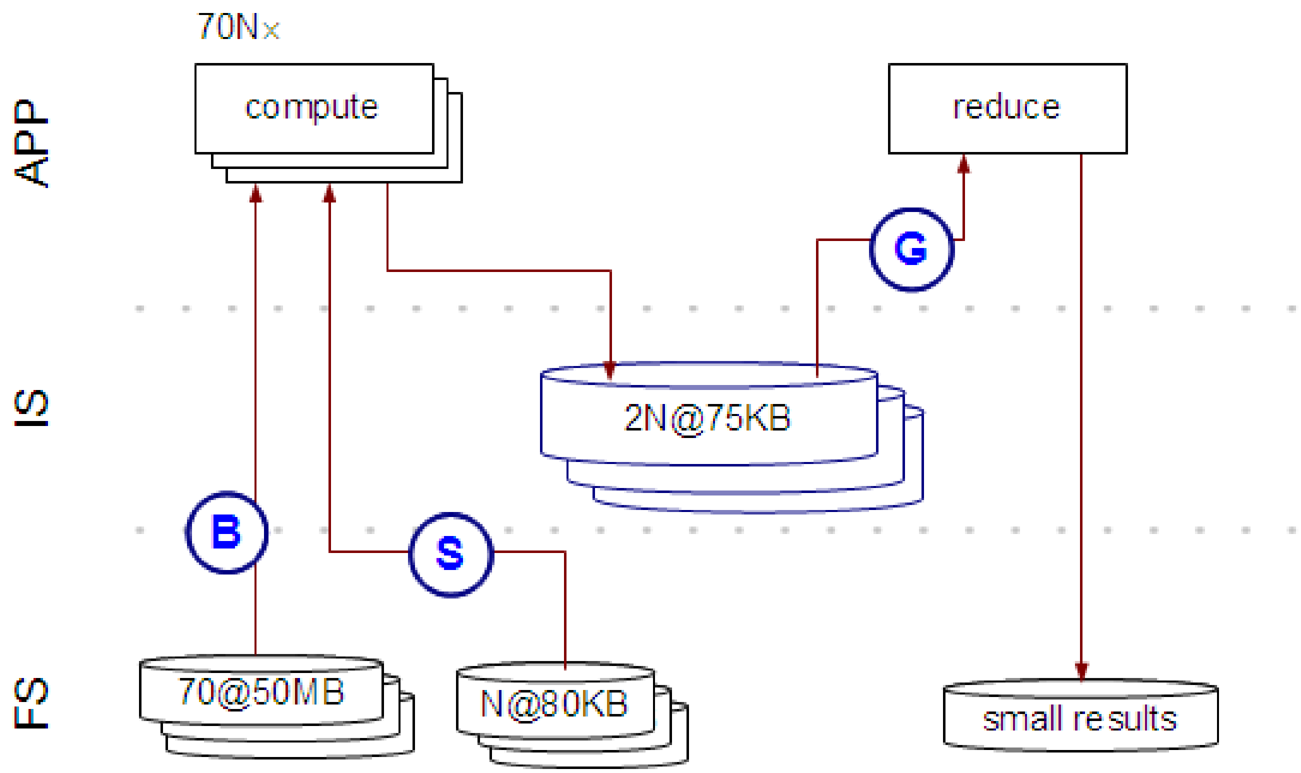
- OOPS: Open Protein Simulator
- DOCK: Molecular docking
- BLAST: Basic Local Alignment Search Tool
- PTMap: Post-transformational modification analysis
- fMRI: Brain imaging analysis

fMRI



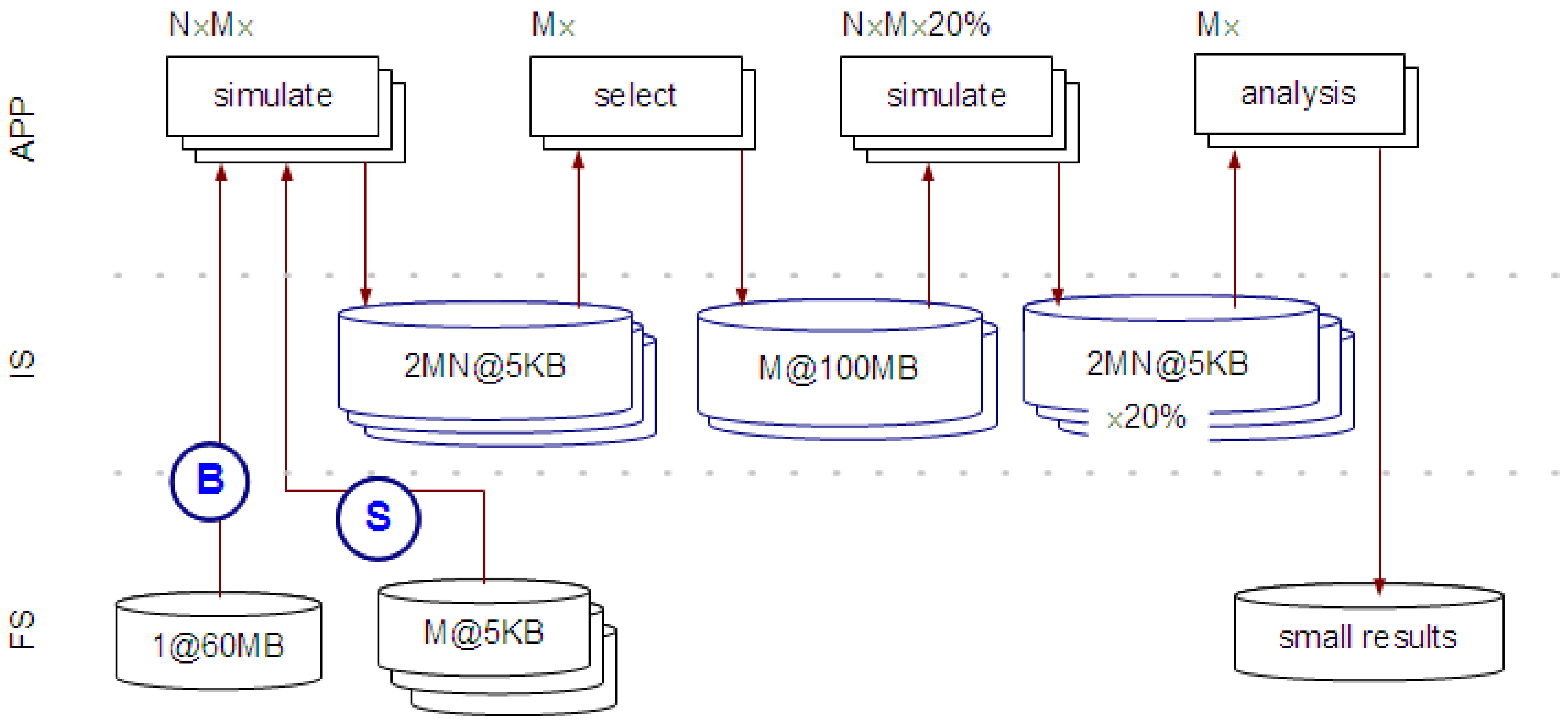
- Simple MapReduce-like structure
- Broken down into **scatter** and **gather** operations
- Intermediate data can be cached. Produces much final output

BLAST



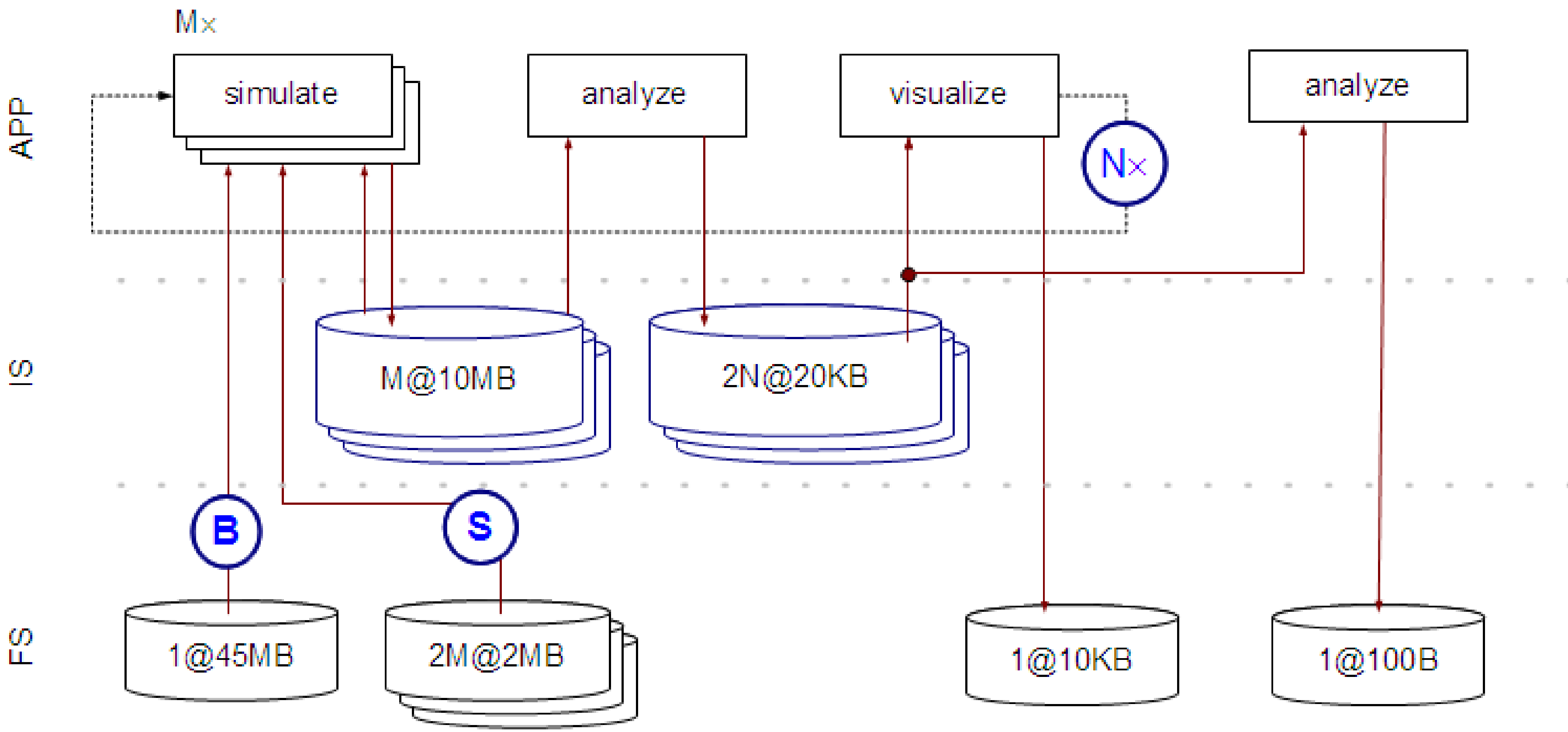
- Like MapReduce with two inputs
- If cache is used to implement **broadcast**, must prevent pollution
- Produces trivial final output – I/O reduction may exceed 99%

DOCK



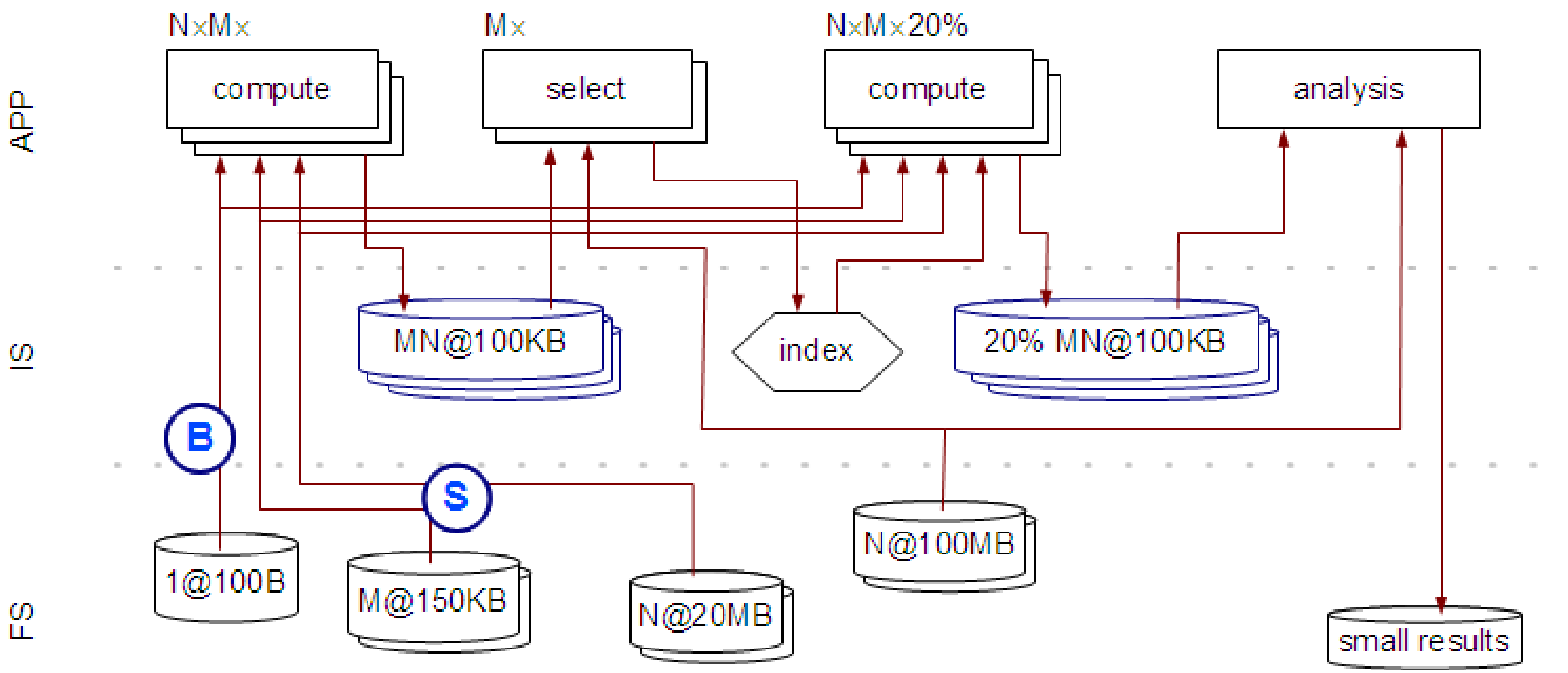
- Significant input size
- Pipeline-like accesses
- Produces trivial final output – I/O reduction may exceed 99%

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

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

- Application collaborators:
  Aashish Adhikari (OOPS) and Sarah Kenny (fMRI)
- Mihael Hategan (Swift, Coasters, Java CoG), Allan Espinosa (BLAST), Ioan Raicu (Falkon)
- Rob Ross
- Thanks to the reviewers
- 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