Argonne NATIONAL LABORATORY

Reliable MPI-IO through Layout-Aware Replication

Seung Woo Son, Samuel Lang, Robert Latham, Robert Ross, Rajeev Thakur
Mathematics and Computer Science Division
Argonne National Laboratory

7th IEEE International Workshop on Storage Network Architecture and Parallel I/O (SNAPI'11)

U.S. DEPARTMENT OF ENERGY

Potential exascale system architecture: everything must scale with compute!

Systems	2009	2018	Difference
System Peak	2 Pflop/sec	1 Eflop/sec	O(1000)
Power	6 Mwatt	20 Mwatt	
System Memory	0.3 Pbytes	32-64 Pbytes	O(100)
Node Compute	125 Gflop/sec	1-15 Tflop/sec	O(10-100)
Node Memory BW	25 Gbytes/sec	2-4 Tbytes/sec	O(100)
Node Concurrency	12	O(1-10K)	O(100-1000)
Total Node Interconnect BW	3.5 Gbytes/sec	200-400 Gbytes/sec	O(100)
System Size (Nodes)	18,700	O(100,000-1M)	O(10-100)
Total Concurrency	225,000	O(1 billion)	O(10,000)
Storage	15 Pbytes	500-1000 Pbytes	O(10-100)
I/O	0.2 Tbytes/sec	60 Tbytes/sec	O(100)
MTTI	Days	O(1 day)	

Source: J. Dongarra, "Impact of Architecture and Technology for Extreme Scale on Software and Algorithm Design," Cross-cutting Technologies for Computing at the Exascale, February 2-5, 2010.

Extreme scale storage systems will be operating in a faulty environment

Checkpoint/restart is key component to achieve fault tolerance in HPC systems

Meeting rapidly growing need for disk bandwidth requires more number of disks (at 65% AGR) because disk bandwidth is growing at 20% AGR

.....

Hard disk failure rate is much higher (by a factor of about 15 times higher) than that expected based on MTTF information supplied by manufactures. (Schroeder and Gibson [FAST'07])

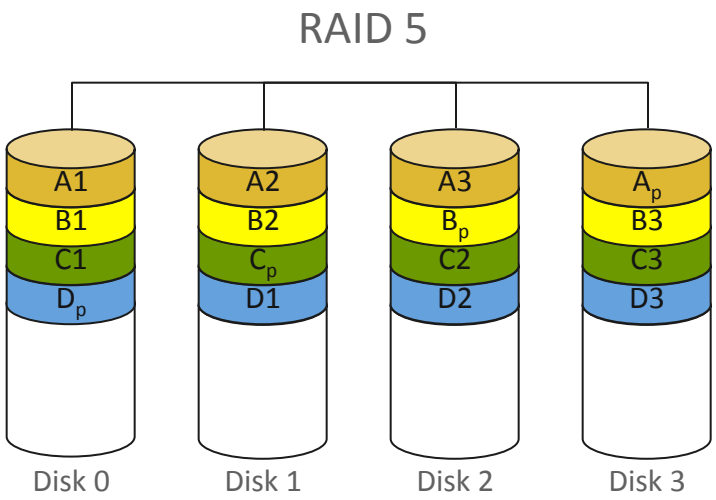
Component failures (MTTF) across several Google cells (Ford et al. [OSDI'10])

Disk	Node	Rack
10~50 years	4.3 months	10.2 yrs

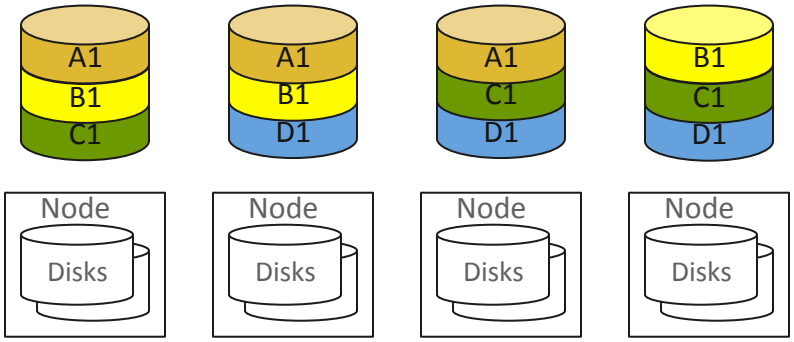
Outline

- Motivation
  - Exascale systems and reliability concern
- Prior work
  - Erasure codes vs. Replication
  - HDFS
- Block replication in MPI-IO
  - Overview
  - Layout-aware MPI Datatypes
  - Client-driven block replication
- Experimental evaluation
  - Microbenchmark and real application benchmark
- Discussion
  - limitations and possible solutions
- Related work
- Conclusion

Erasure codes vs. replication



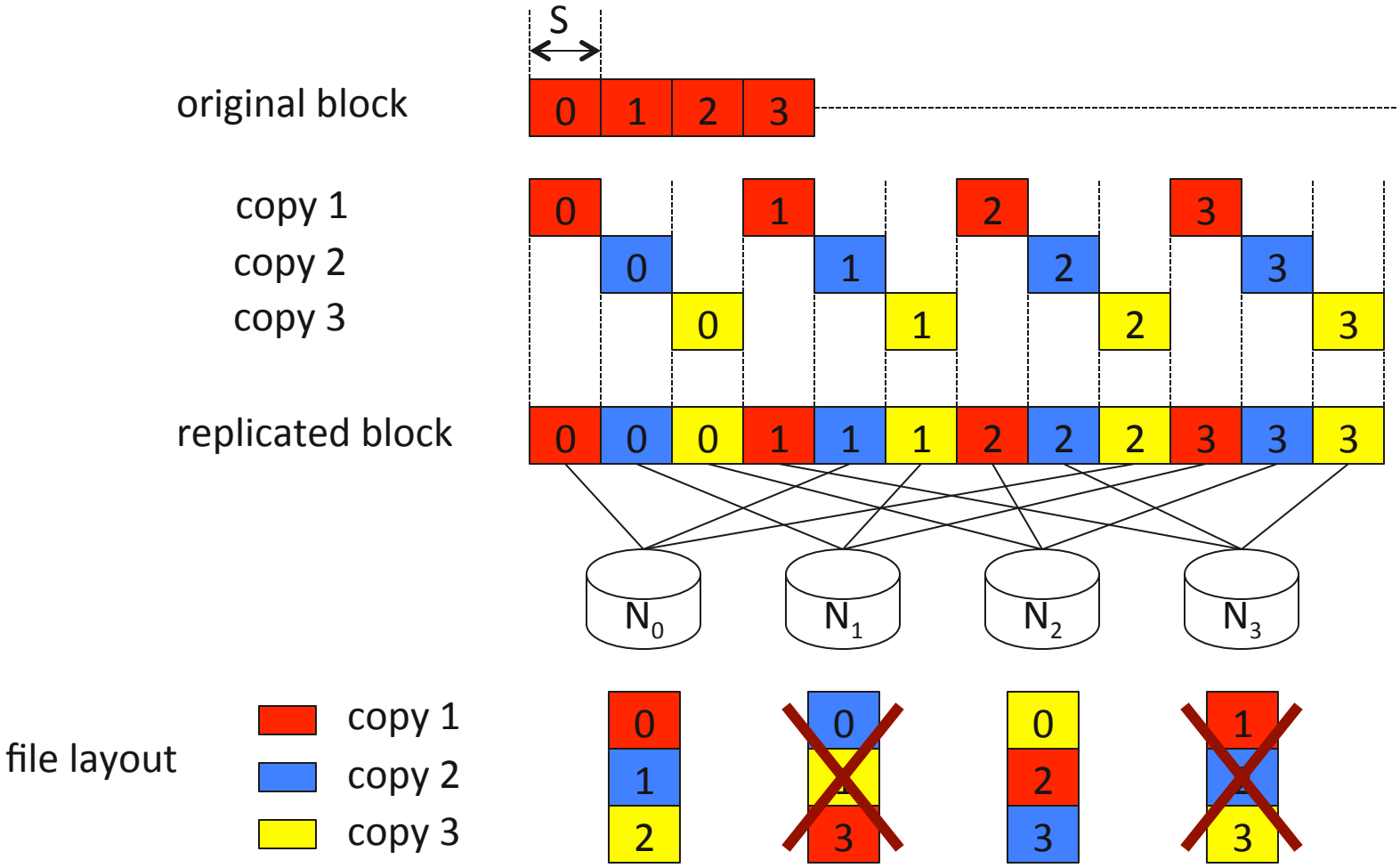
- Typically underneath the parallel file systems
- Works at a fine block granularity
- High cost to purchase hardware disk arrays
- Needs to read two or more of the remaining blocks for rebuilding



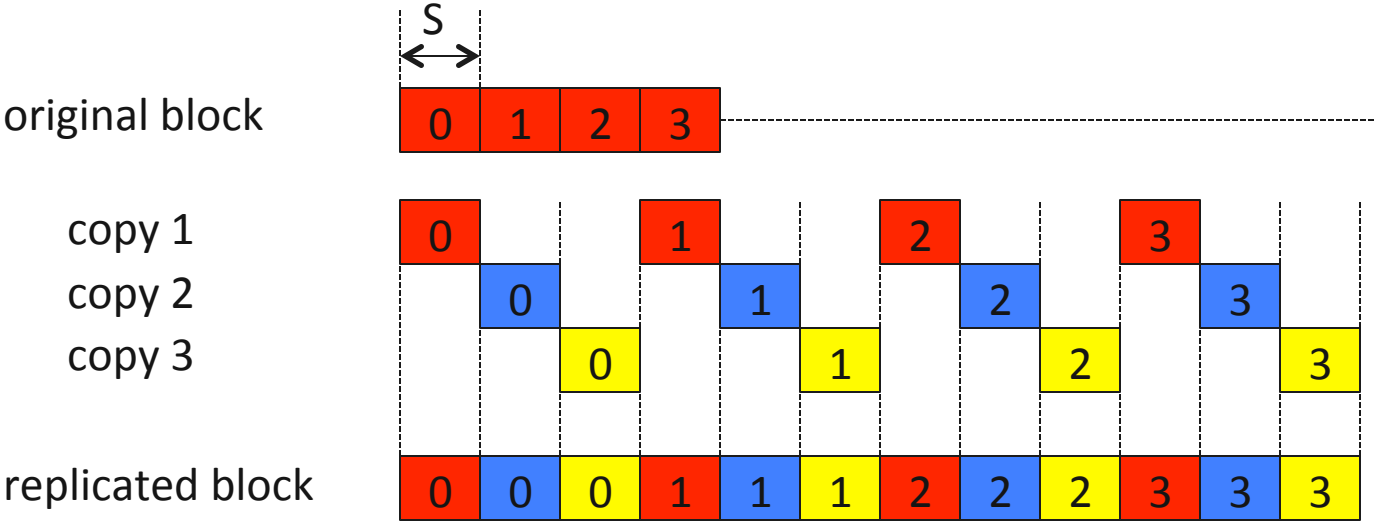
- Recently used in distributed file systems for MapReduce/Hadoop, e.g., HDFS, GFS
- Coarse block granularity, 64MB in HDFS
- Requires only one copy for rebuilding
- High storage overhead, e.g., 200%

We want to emulate replication within MPI-IO!

Block replication in a single file across stripe boundaries



We use MPI datatypes for representing each replica



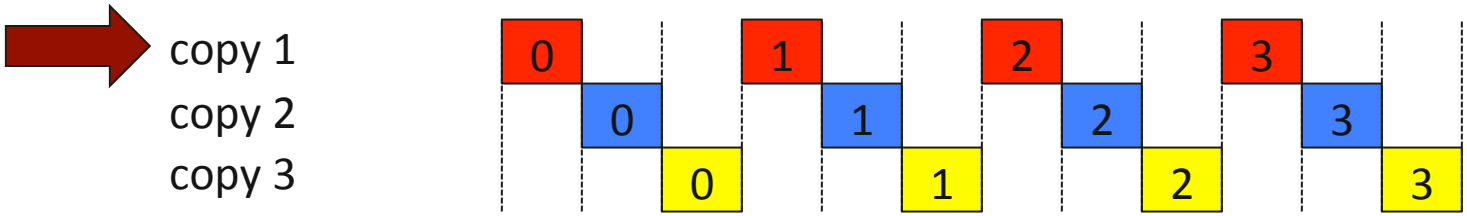
$$D_{orig} = \{(s_0, d_0), (s_1, d_1), \cdots, (s_{n-1}, d_{n-1})\}$$

$$D_{copy1} = \{(s_0, d_0), (s_1, d_1 + 3S), \cdots, (s_{n-1}, d_{n-1} + 3S * (n-1))\}$$

$$D_{copy2} = \{(s_0, d_0 + S), (s_1, d_1 + S + 3S), \cdots, (s_{n-1}, d_{n-1} + S + 3S * (n-1))\}$$

$$D_{copy3} = \{(s_0, d_0 + 2S), (s_1, d_1 + 2S + 3S), \cdots, (s_{n-1}, d_{n-1} + 2S + 3S * (n-1))\}$$

MPI derived dataypes and fileview



MPI-IO allows users to access several noncontiguous pieces of data by defining file views with derived datatypes.

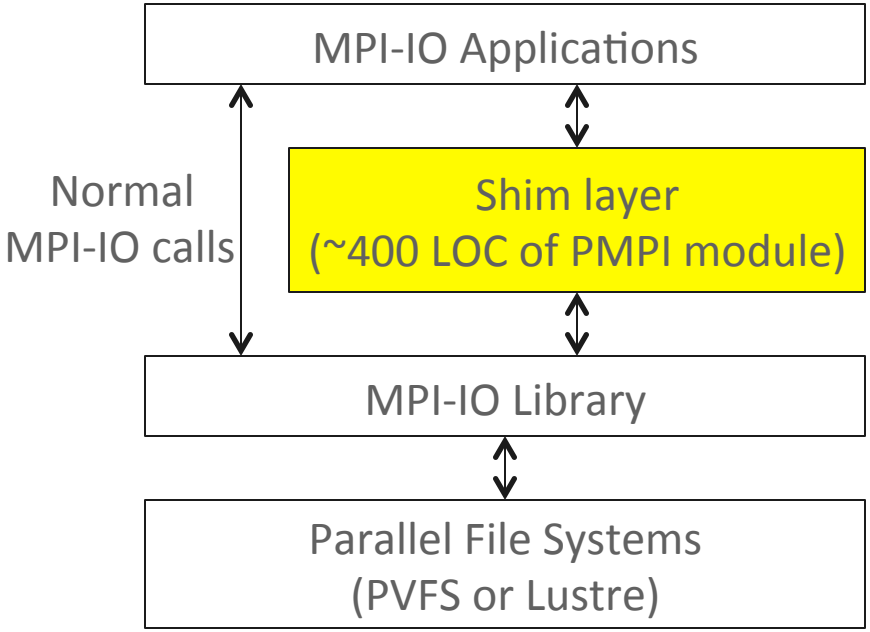
$$D_{copy1} = \{(s_0, d_0), (s_1, d_1 + 3S), \cdots, (s_{n-1}, d_{n-1} + 3S * (n-1))\}$$

```
ncount = 4;
blocklens[ncount] = {S, S, S, S};
offsets[ncount] = {0, 3S, 6S, 9S};

MPI_Type_create_hindexed (ncount, blocklens[], offsets[],
                          old_type, &newtype);
MPI_Type_commit (&newtype);


MPI_File_set_view (fh, 0, old_type, newtype, …);
MPI_File_write (fh, buf, count, old_type, &status);
```

Replication is done transparently

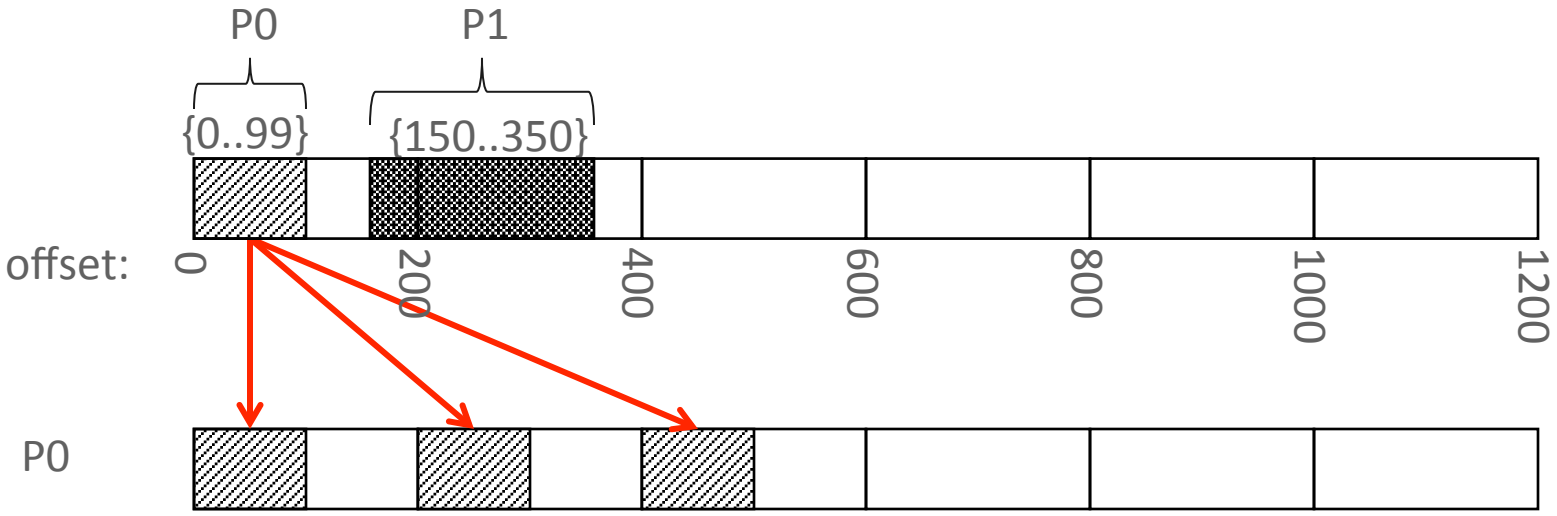



```
MPI_Info_set (info, "RF", "3");
MPI_File_write(fh, buf, count,
                   MPI_INT, &status);
```

```
MPI_File_write(…)
{
  create 3 derived datatypes;
  for(i=0; i<RF; i++)
  {
    MPI_File_set_view(…);
    PMPI_File_write(…);
  }
}
```

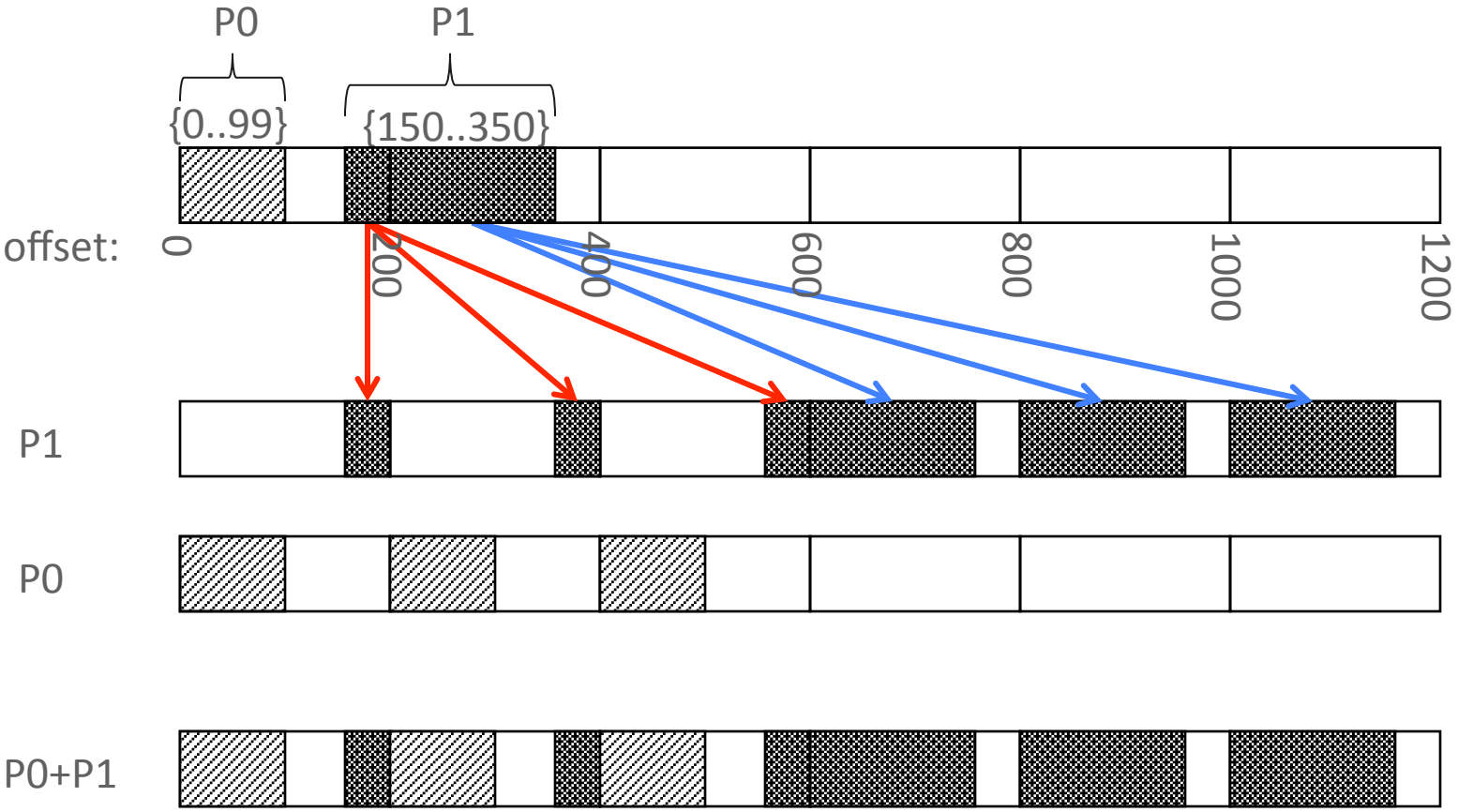
Example

Assuming stripe size = 200 bytes,
Rank 0 (P0) writes 100 bytes at offset 0, and
Rank 1 (P1) writes 200 bytes at offset 150

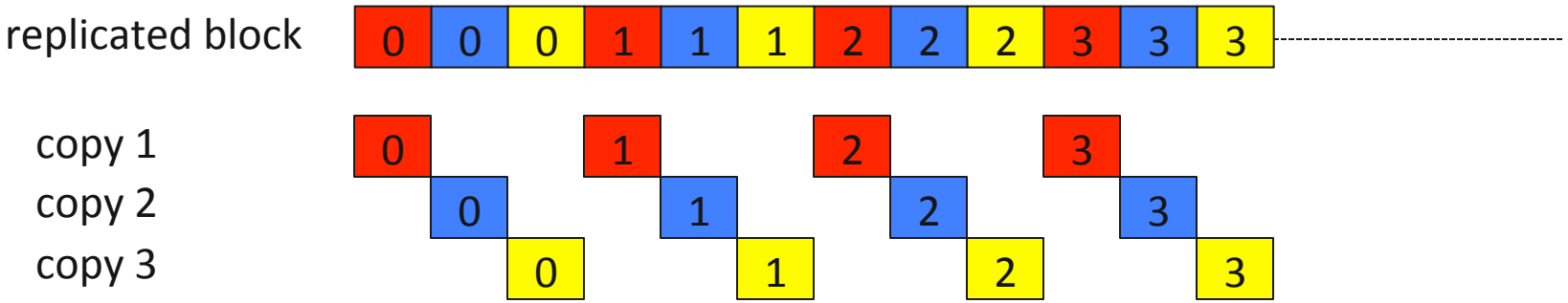


Example - cont'd

Assuming stripe size = 200 bytes,
Rank 0 (P0) writes 100 bytes at offset 0, and
Rank 1 (P1) writes 200 bytes at offset 150



How to deal with reads?

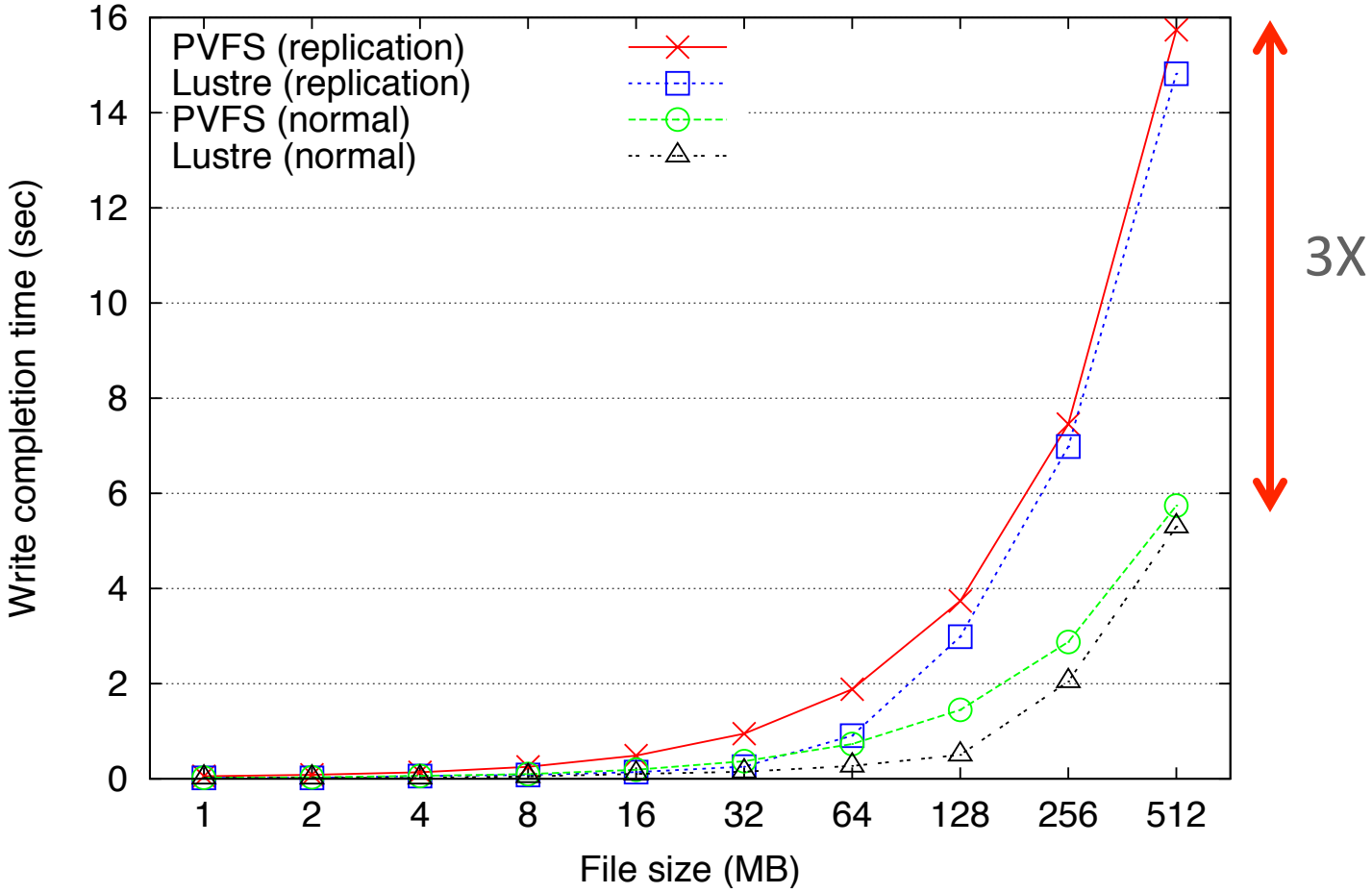



```
MPI_File_read(...)
{
  /* try copy 1 */
  MPI_File_set_view(fh_shadow, 0, ..., hindextype[0], ...);
  PMPI_File_read(fh_shadow, buf, nints, MPI_INT, &status);
  if (status == ERROR)
  {
    /* try copy 2 */
    MPI_File_set_view(fh_shadow, 0, ..., hindextype[1], ...);
    PMPI_File_read(fh_shadow, buf, nints, MPI_INT, &status);
    if (status == ERROR) {
      /* try copy 3 (last) */
      MPI_File_set_view(fh_shadow, 0, ..., hindextype[2], ...);
      PMPI_File_read(fh_shadow, buf, nints, MPI_INT, &status);
    }
  }
}
```

Experimental Methodology

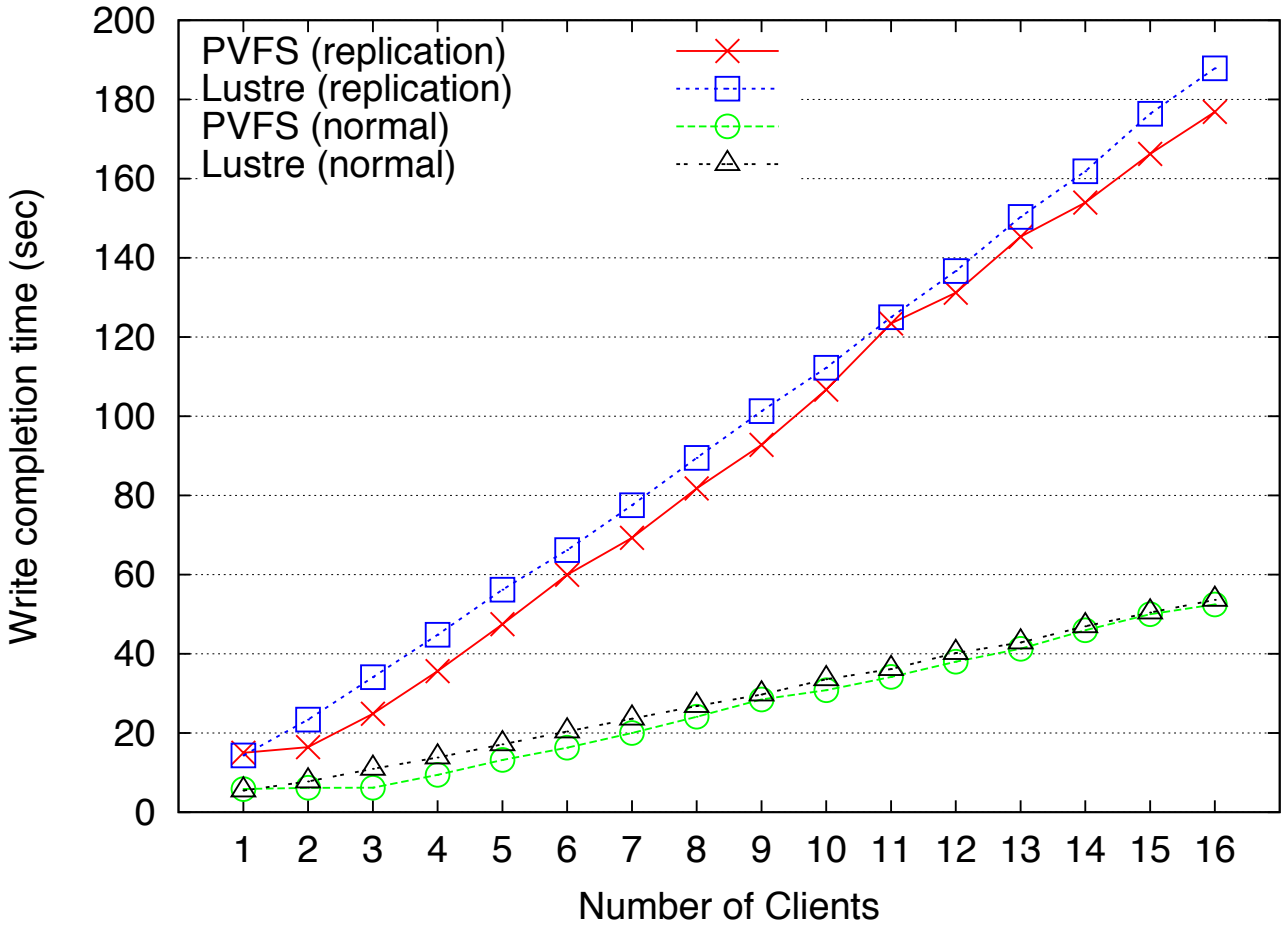
- Evaluation platform
  - A cluster of 24 nodes
    - Each node: Dual Intel Xeon Quad core 2.66 GHz, 16 GB main memory, 50 GB local storage space
  - All nodes run the Linux 2.6.22 kernel, connected thorough 1GE
  - MPI library: MPICH2-1.3.1
  - Parallel file systems: PVFS-2.8.1 and Lustre 1.6.4.2
    - Configured to use 4 storage nodes
    - Default stripe size: 1 MB
- Evaluated schemes
  - Normal: without replication
  - Replication: our MPI triplication scheme
- Evaluation metric
  - Write completion time: micro and application benchmark

Replication overhead increases w.r.t replication factor

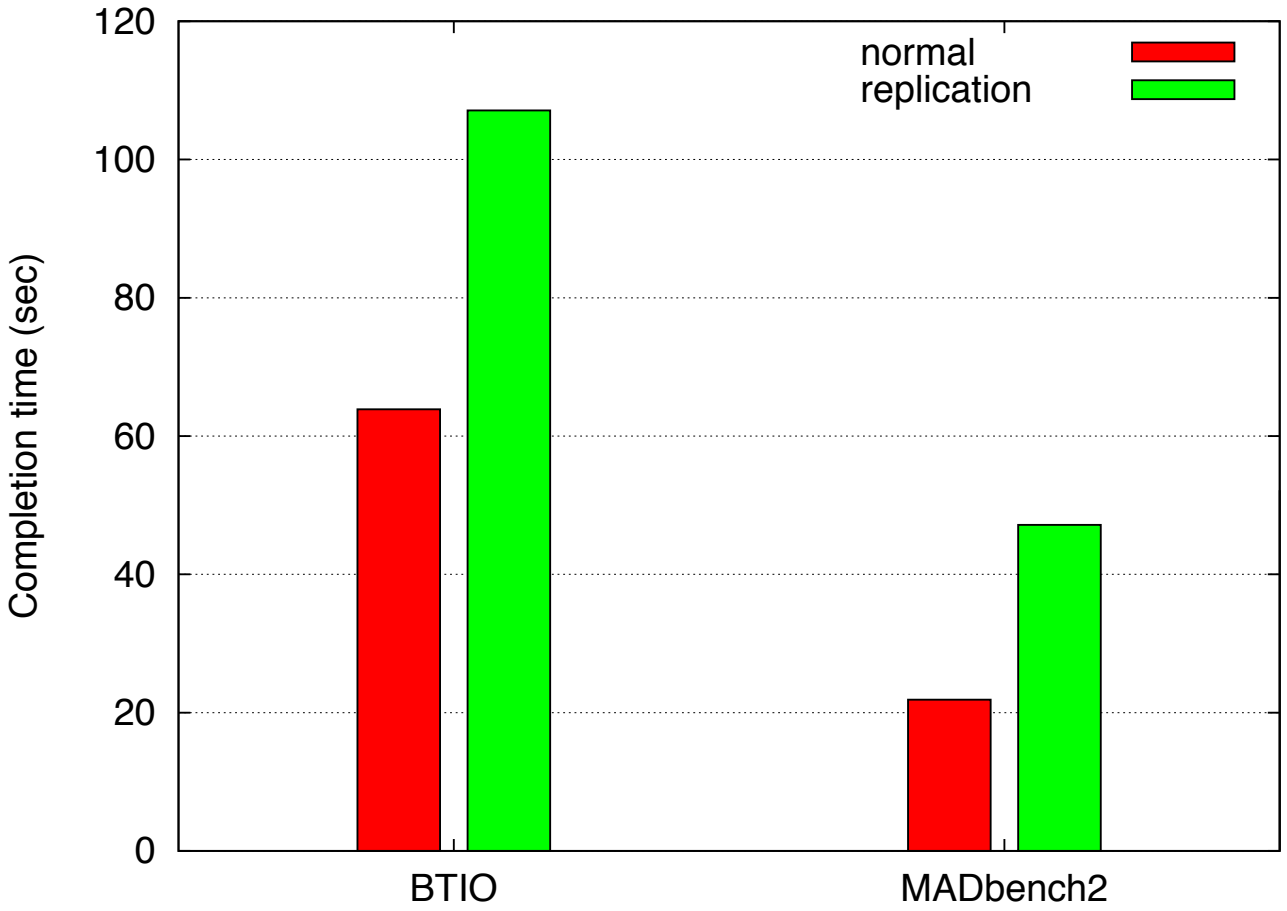


write completion time $\propto$ replication factor (3)

Our replication scheme is scalable w.r.t number of clients (writers)



Real applications

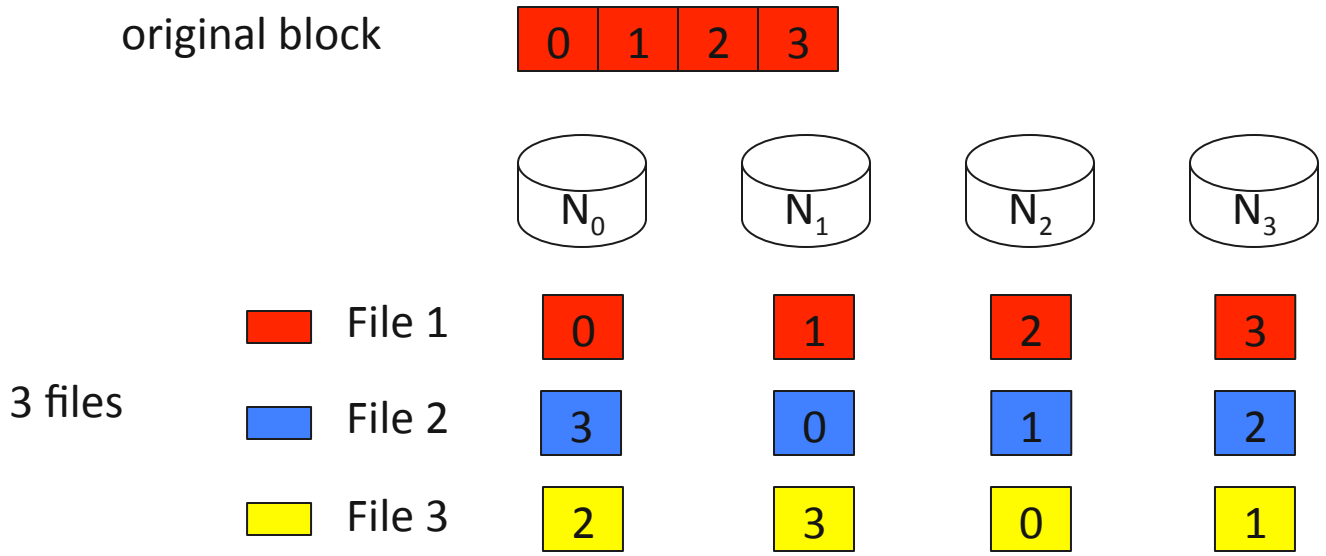


67.7% and 115.6% performance penalty for BTIO and MADbench2, respectively

Discussion

- Collective I/O
  - Current implementation works only with independent calls
    - MPI_File_write(), MPI_File_write_at(), etc.
  - Fileview conflict
    - Fileview for describing different block location ≠ Fileview for a particular process
- Storage overhead
  - Higher storage overhead than RAID
    - 25% in 8+2 RAID 6 vs. 200 %
  - Selective replication or async/delayed encoding can be used in conjunction with
- Non-blocking writes for replicas
  - Currently ROMIO does not support nonblocking I/O for strided writes
- Placing replicas in separate files
  - Replication can be done in a single file or multiple files

Example: file layouts using three files with different first data servers



It achieves the same reliability level, i.e., tolerating two disk (node) failures.

Each file stripe should start at a different file server (disk).

It does not require complex data layouts for read/write calls.

It creates 3x more files than single-file approach.

Related work

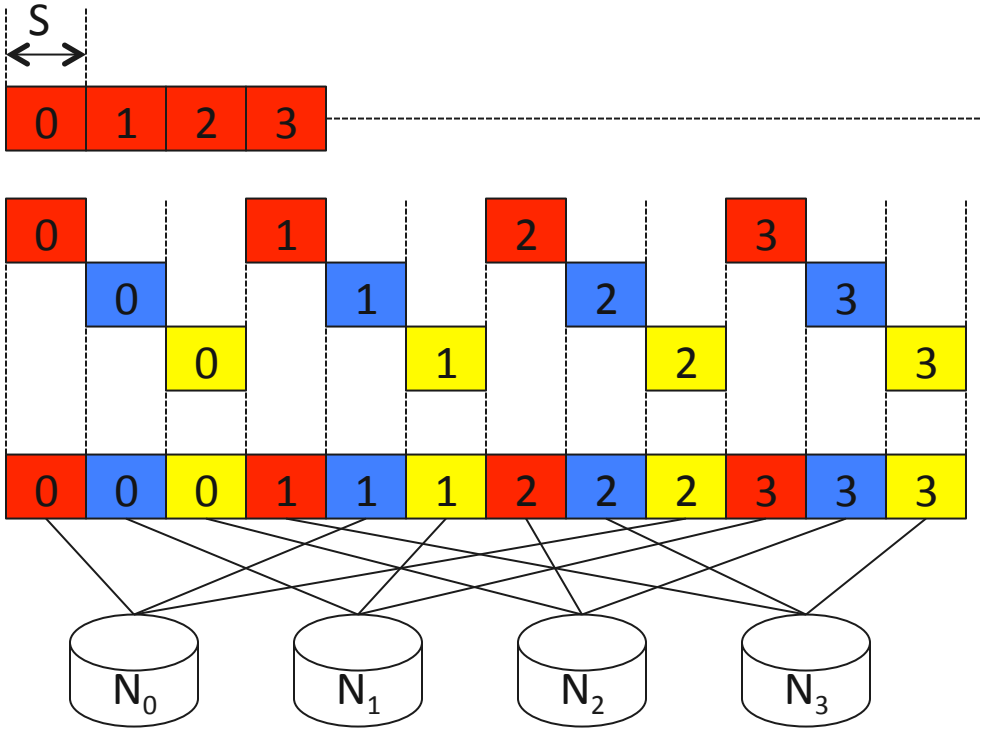
- MPI extensions for fault tolerance
  - Most are focused on providing capability of checkpointing/restart
  - FT-MPI, VolpexMPI, XOR-based double erasure codes [Wang et al.], transparent redundancy [Brightwell et al.]
  - Not providing redundancy to data stored on the storage systems
- Redundancy in MPI-IO and parallel file systems
  - Fault tolerance in MPI applications on PVFS [Calderon et al.], Data structure for continuous snapshot [Brinkmann et al.], SSpiRAL [Amer et al.]
- Data-intensive computing workloads, e.g., MapReduce/Hadoop
  - HDFS uses triplication for fault tolerance
  - Our approach is similar, but provides replication within the context of MPI-IO
- Lazy redundancy [Gropp et al.]
  - It also uses MPI datatypes to calculate parity blocks
  - It requires a modification to the ROMIO MPI-IO implementation

We provide a block replication within MPI-IO transparently

We use MPI derived datatypes to represent replicated file layouts.

Block replication is implemented within shim layer.

Our scheme can be used with any existing parallel file systems.



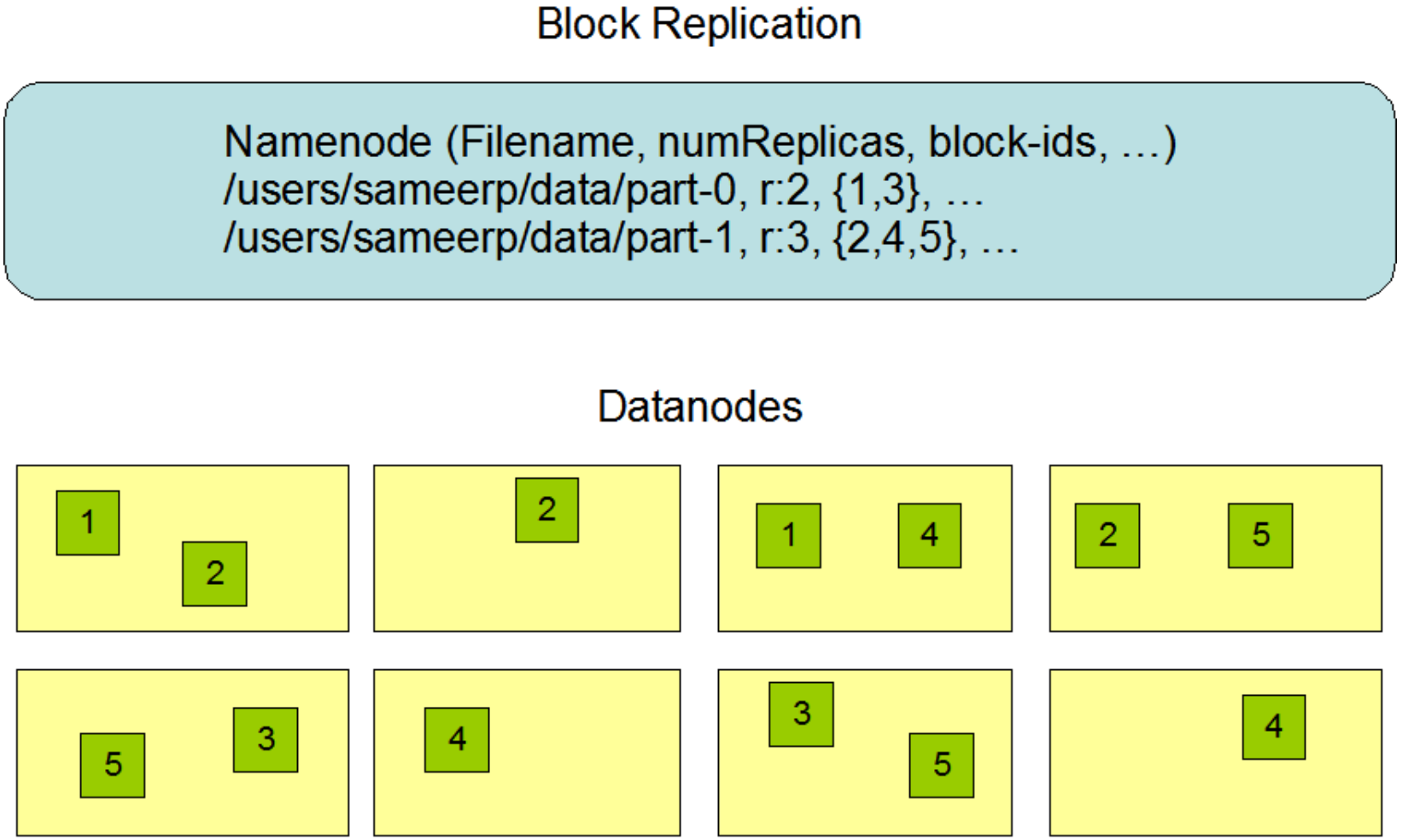
Acknowledgements

- DOE Office of Advanced Scientific Computing Research (ASCR)
- Wei-keng Liao (NU)

Questions?

Backup slides

HDFS replicates block in a rack-aware manner



Source: http://hadoop.apache.org/common/docs/r0.18.0/hdfs_design.pdf

Application Benchmarks

Benchmark	Description	No. of Procs	Exec time	Dataset size
BTIO	- I/O version of BT (block tridiagonal) - "simple" subtype with Class A problem size (64 x 64 x 64 grid size) - 419.43 MB data to a shared file every 5th timestep out of 200 iterations	16	69.9 sec	419.43 MB
MADbench2	- MADspec data analysis code - Out-of-core matric operations - Built to generate unique (individual) file type using MPI-IO - Each process writes about 303 MB	16	21.9 sec	4,848 MB