
# On implementing MPI-IO on BlobSeer

Viet-Trung TRAN
KerData Team

# Common I/O needs


- Multiple I/O needs of scientific applications:
  - Reading initial input
  - Writing results
  - Checkpointing
  - Out-of-core computations
  - Visualization

# I/O access patterns in parallel applications


- Different from those in sequential programs
- Sequential programs typically access data in large, contiguous chunks
- In many parallel programs, each process may need to access several noncontiguous pieces of data from a file
- Group of processes may access the file simultaneously, and the accesses of different processes may be interleaved in the file

# Need for parallel I/O


- As computers get larger and faster, I/O becomes even more a bottleneck
- Sequential I/O becomes a bottleneck
- Standard unix I/O is not portable nor suitable
  - Express access patterns with a single function call
  - No notion of collective I/O
  - Vendor-specific extensions to Unix API is nonportable
- True parallel I/O requires multi-process access from a parallel file system

# Advantages of MPI I/O for parallel I/O


- MPI I/O can be considered as Unix I/O plus (lots of) other stuff
- Writing is like sending a message and reading is like receiving
  - Performance
  - Portability
  - Convenience

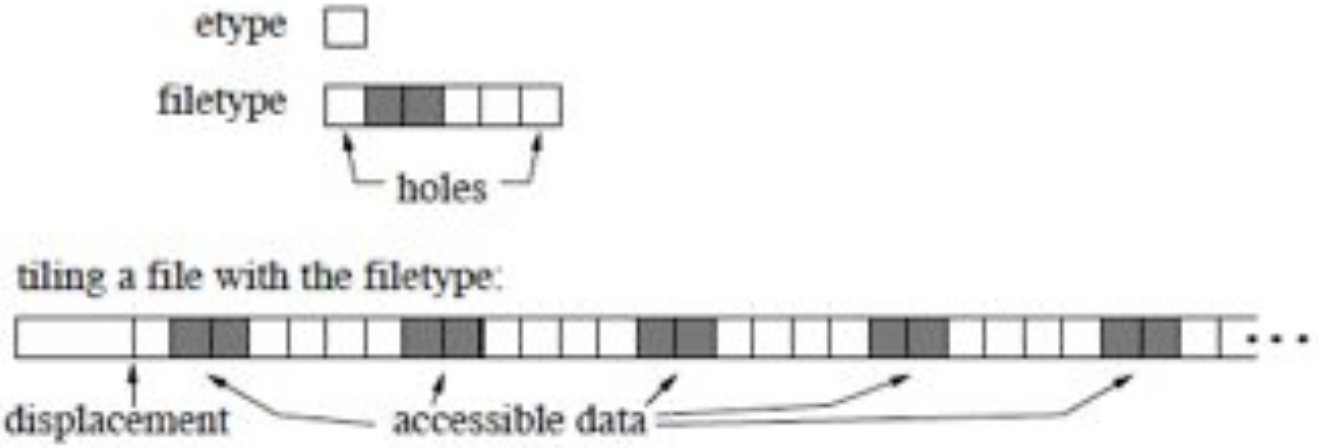
# Features of MPI I/O


- Parallel read/write
- Non-contiguous data read/write
- Non-blocking read/write
- Collective read/write
- Portable data representation across platforms
- Mechanism for providing hints applicable to a particular storage system

# Fileview


- Displacement, etype and filetype creates a fileview
- fileview allows simultaneous writing/reading of noncontiguous interleaved data by multiple processes
- MPI_File_set_view call
- each process has a different fileview of a single file

# Fileview example



# Fileview example (2)

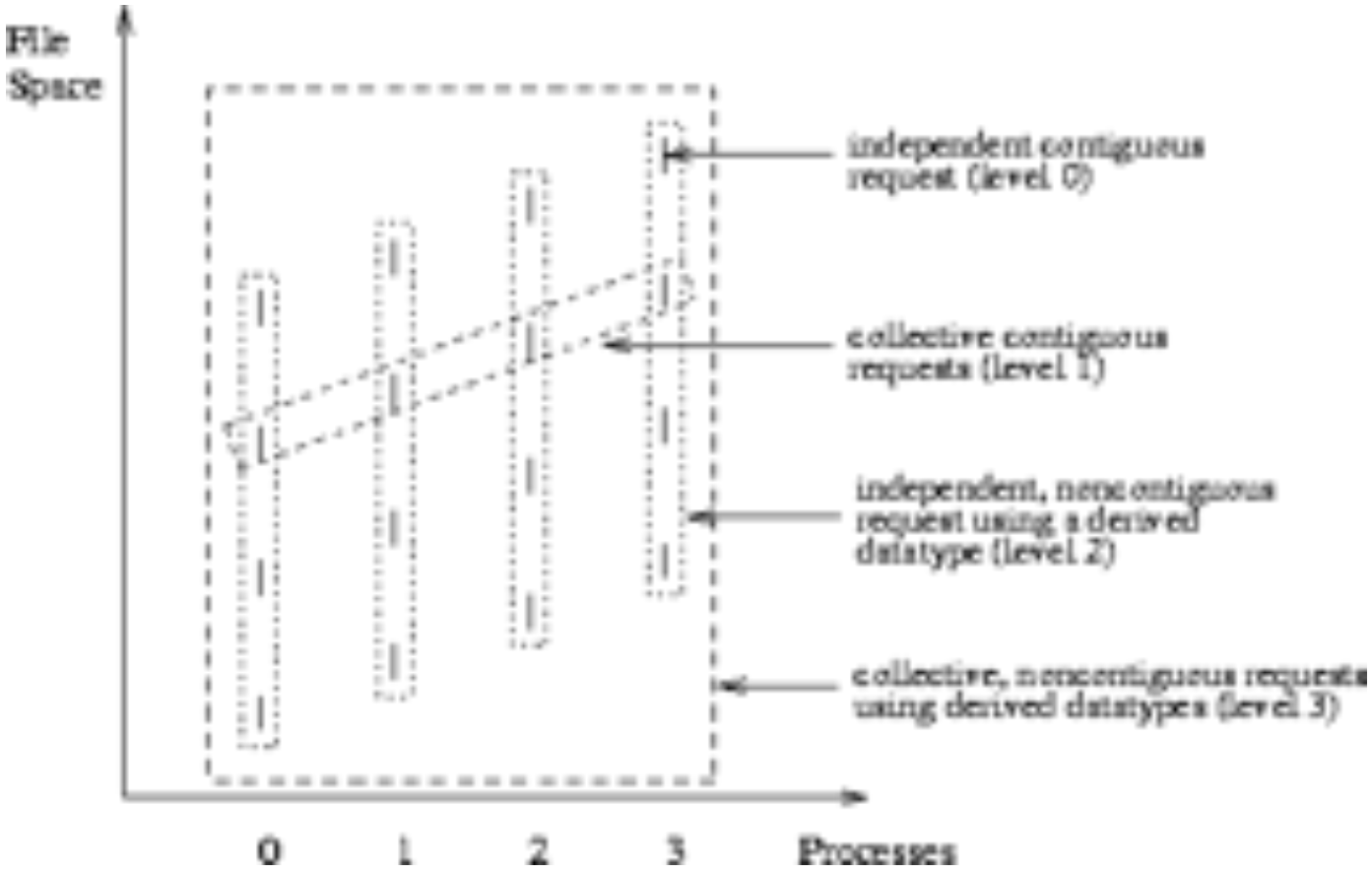

- Using fileviews to achieve a global data distribution


Figure 13.2: Partitioning a file among parallel processes

# Collective I/O

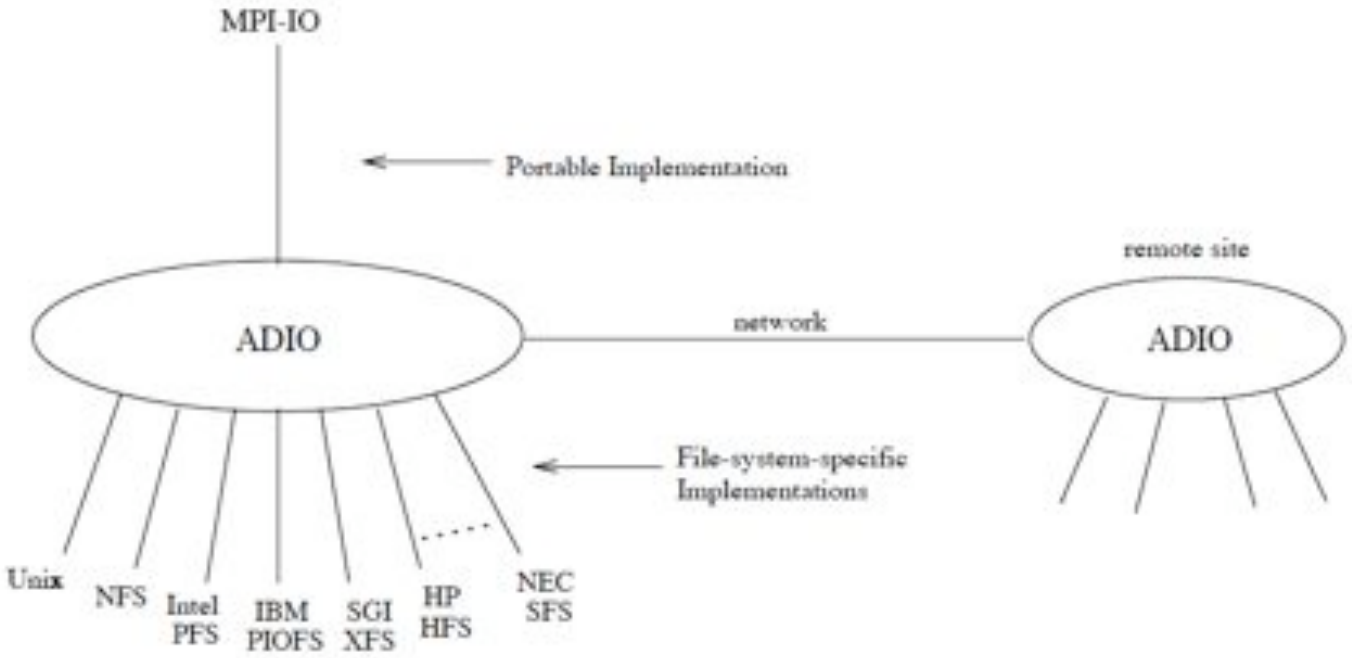

- Optimized I/O based on combined requests of all processes
- Can merge the requests of different processes to perform I/O efficiently
- Good for small/noncontiguous requests

# 4 levels of MPI-I/O



# ROMIO: a MPI-IO implementation


- High performance, portable



# Parallel file system's features needed


*Table 1.* parallel file system's features needed.

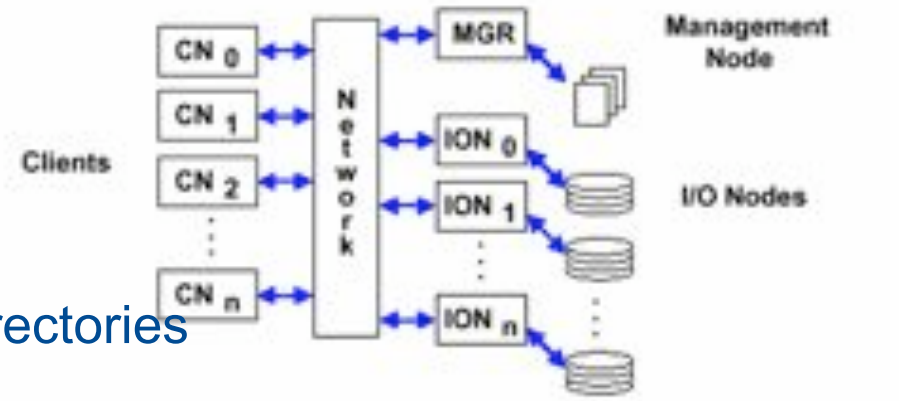
| Features | MPI-IO | PVFS v2 |
|---|---|---|
| High-Performance Parallel File Access | yes | yes |
| Data-Consistency Semantics | yes | yes |
| Tunable atomicity Semantics | yes | yes |
| Tunable File-Attribute Consistency | yes | yes |
| Interface Supporting Noncontiguous Accesses | yes | yes |
| Support large files | yes | yes |
| Byte-Range Locking | yes | yes |
| Control over File Striping | yes | ? |
| Nonblocking (Asynchronous) I/O Optional | yes | yes |
| Elasticity | ? | no |

# Case study: PVFS


- Parallel virtual file system
- Goals of PVFS
  - Provide *high-speed access* to file data for parallel applications
  - Provide a cluster-wide *consistent name space*
  - Enables *user-controlled striping of data* across disks on different I/O nodes

# PVFSv1 (2000)


- Function of each component
  - Metadata Server
    - One per PVFS file system
    - Maintains information on files and directories
  - I/O Servers
    - More than one nodes
    - Store PVFS file data on local file systems such as ext2fs partition
  - Clients
    - Users of the PVFS system
    - Applications accessing PVFS files and directories run on client machines



# PVFSv1


- Metadata server needs not to be involved in I/O operations

Table 1: Metadata example: File /pvfs/foo.

| | |
|---|---|
| inode | 1092157504 |
| : | : |
| base | 2 |
| pcount | 3 |
| ssize | 65536 |

# PVFSv1: acknowledged inconveniences


- Single metadata server
  - Single point of failures
  - Bottleneck for even well-behaved applications
- Fixed data distribution in round-roubin fashion
  - Can not benefit from access patterns

# PVFSv2


- Distributed metadata
- PVFSv2 server can be I/O node, Metadata node or both
- Structured non-contiguous data access
- flexible and extensible data distribution
- tight MPI-IO integration
- Explicit support for concurrency
- Tunable semantics
- Data and metadata redundancy

# Parallel file system's features needed


*Table 1.* parallel file system's features needed.

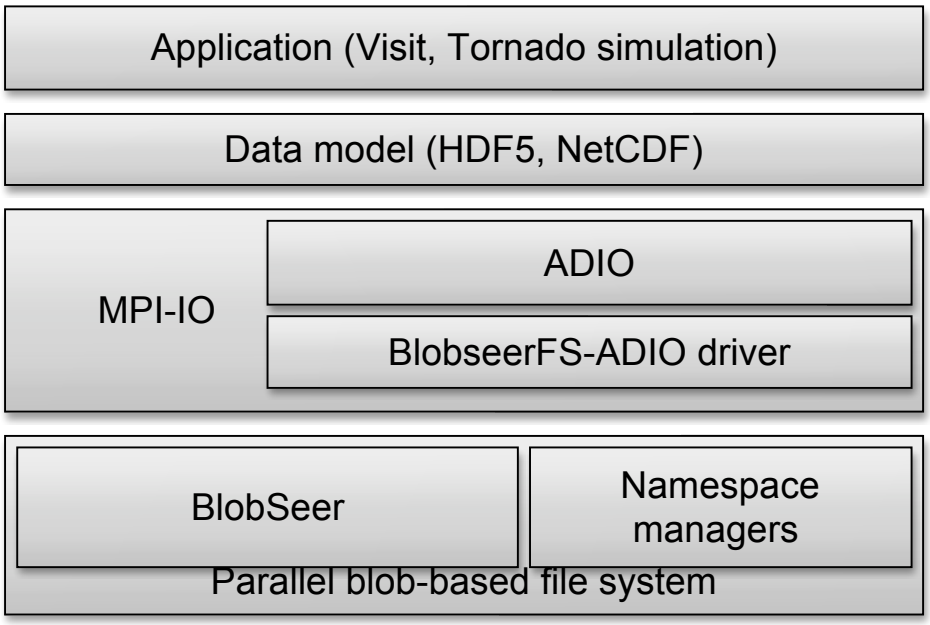
| Features | MPI-IO | PVFS v2 |
|---|---|---|
| High-Performance Parallel File Access | yes | yes |
| Data-Consistency Semantics | yes | yes |
| Tunable atomicity Semantics | yes | yes |
| Tunable File-Attribute Consistency | yes | yes |
| Interface Supporting Noncontiguous Accesses | yes | yes |
| Support large files | yes | yes |
| Byte-Range Locking | yes | yes |
| Control over File Striping | yes | ? |
| Nonblocking (Asynchronous) I/O Optional | yes | yes |
| Elasticity | ? | no |

# BlobSeer vs PVFSv2 wrt. MPI-IO


| Feature | PVFSv2 | BlobSeer |
|---|---|---|
| Tolerance to single point of failures | YES | Not yet |
| Elasticity | ? | ? |
| Distributed metadata | YES | A part |
| Tight MPI-IO coupling | YES | Not yet |
| Concurrency | YES | YES |
| Checkpointing | NO | YES |
| Versioning | NO | YES |

# BlobSeer as a parallel file system


- Parallel I/O Software Architecture



# Current status

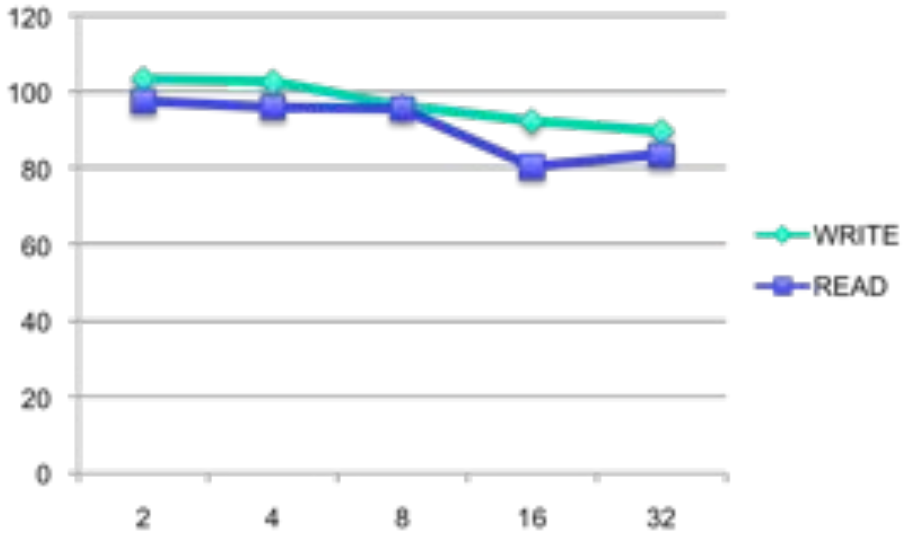

- Current status
  - Coded a simple ADIO layer for ROMIO
  - Optimization for collective IO and noncontiguous IO is based on the default strategies: Data sieving and Two-Phase I/O
  - Tested with MPICH-2 on Grid'5000
  - Got some preliminary results

# PVFS vs BlobSeer


- Experiment on Grid5000
- PVFS is convenient for MPI-IO (64k page size in default)
  - BlobSeer
    - Write bandwidth = 54.498929 Mbytes/sec
    - Read bandwidth = 46.329913 Mbytes/sec
  - PVFS
    - Write bandwidth = 107.976045 Mbytes/sec
    - Read bandwidth = 64.365856 Mbytes/sec
-

# Microbenchmark


- Using mpi-io test included in PVFS for individual IO (each write/read 64 MB)
- Deployed on 32 nodes on the Rennes site of Grid'5000
- Mesure the average throughput (~MB)



# BTIO benchmark


- Using NASA BTIO benchmark, Class B, Collective IO
- Measure the total execution time in seconds