Towards Scalable Application Checkpointing with Parallel File System Delegation

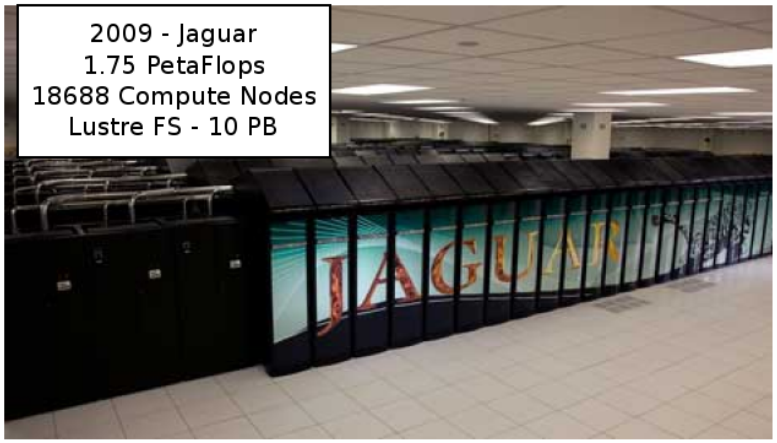
Dulcardo Arteaga darte003@fiu.edu
Ming Zhao ming@cs.fiu.edu

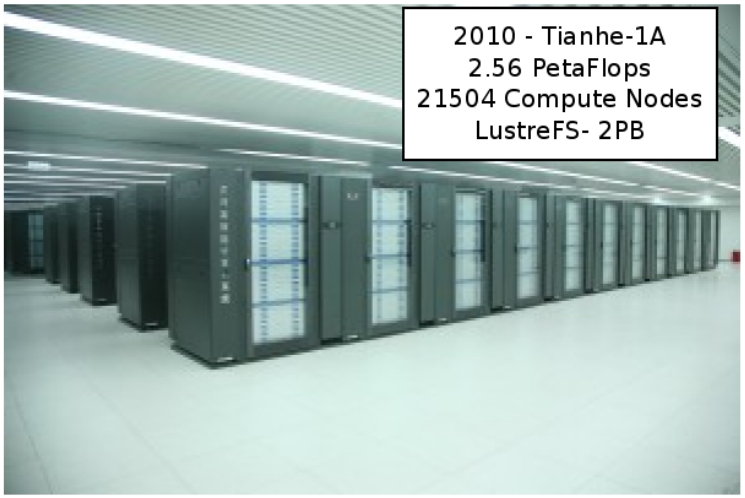
School of Computing and Information Sciences
Florida International University
Miami, FL

VISA

FIU
Florida International University
Miami's public research university

High Performance Computing Systems

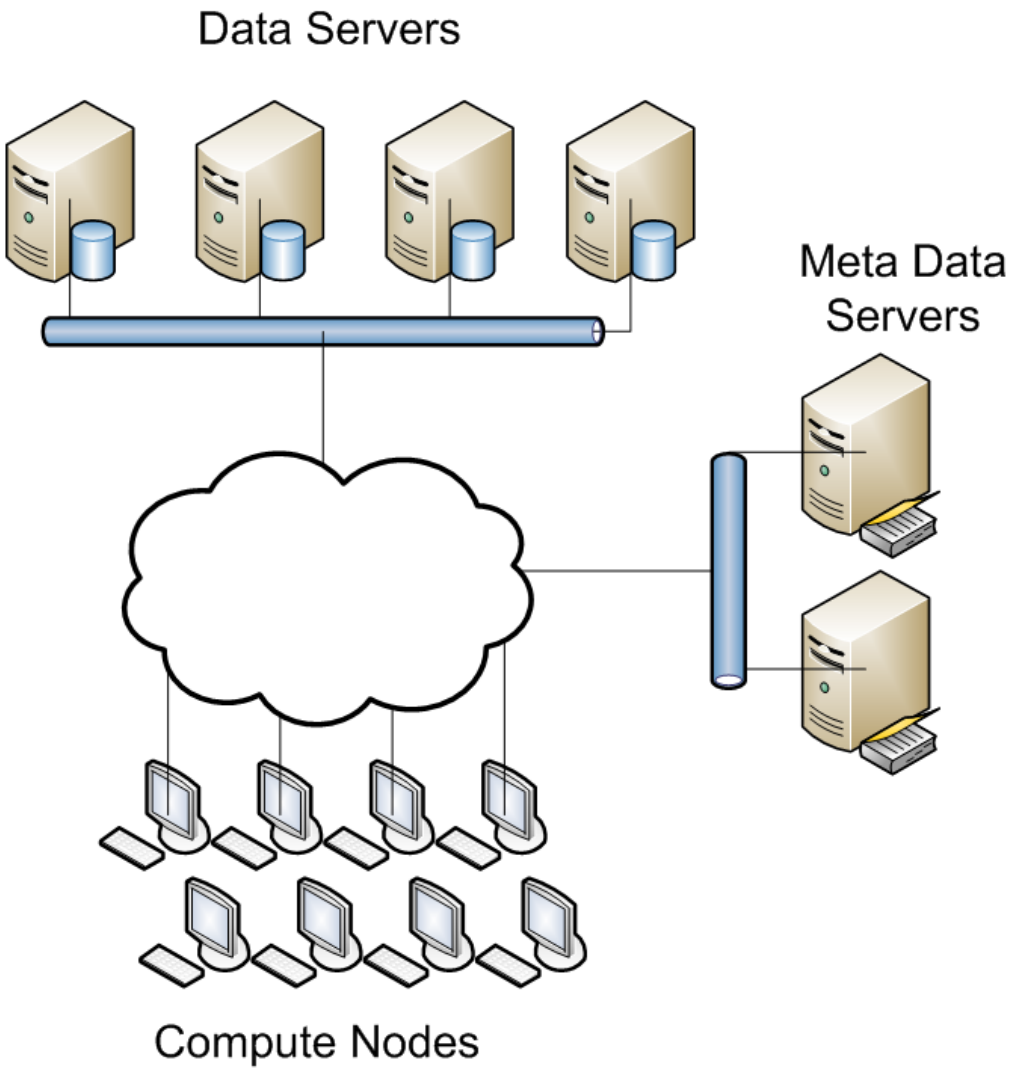




Introduction

- Scalability
  - Large scale applications run on HPC
  - One important challenge is **Fault Tolerance**
  - Common approach is checkpointing
- Checkpointing
  - Store a snapshot of the current application state
  - Applications recover from valid snapshot in case of failure
- HPC systems use parallel file system to do checkpointing

Parallel File Systems (PFSes)



- Components:

Meta Data Servers
Store metadata information about files
Data Servers
Store actual data of files
Clients
Run on compute nodes and provide interface to Storage System

Problem Statement

Problem

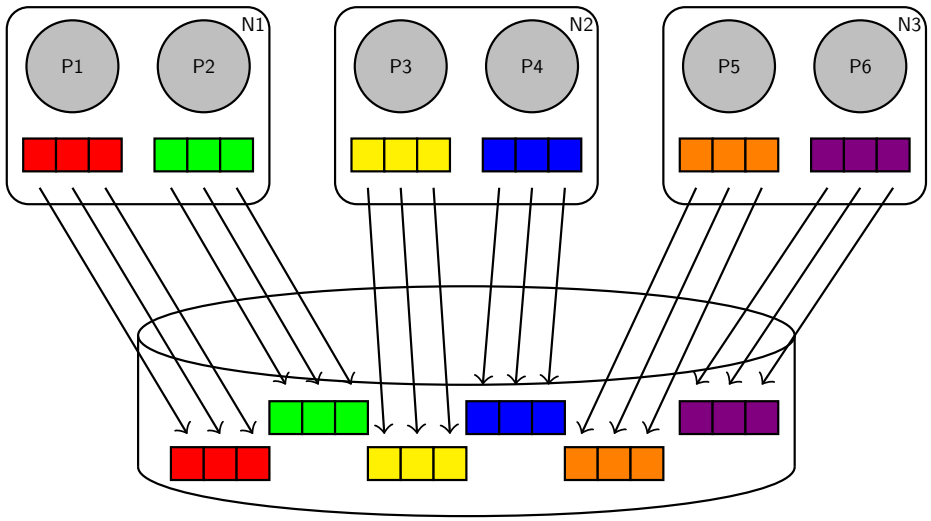
Large scale checkpointing causes serious **bottleneck at metadata servers** on HPC systems

Approach

Delegate the management of the PFS storage space used for checkpointing to applications to reduce metadata overhead

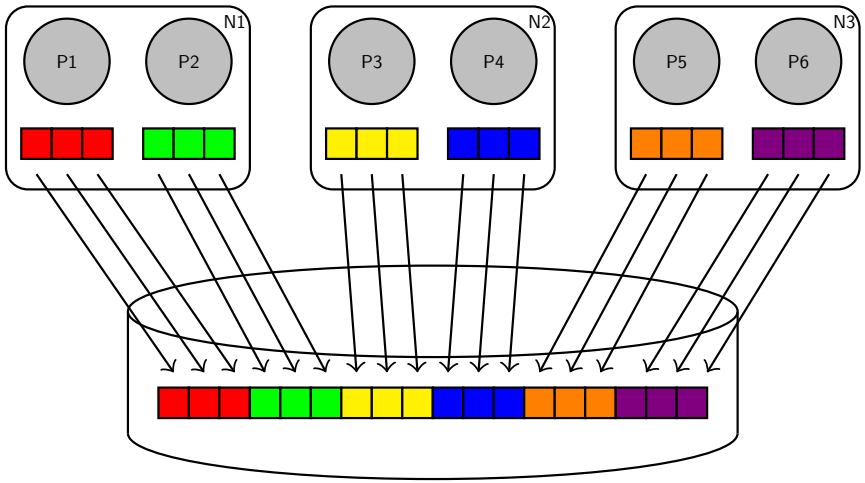
Outline

- File-per-Process (N-N)
  - Every process writes to a different file
- Metadata management overhead
  - Imply a creation of many files
  - Metadata operation per file and per process

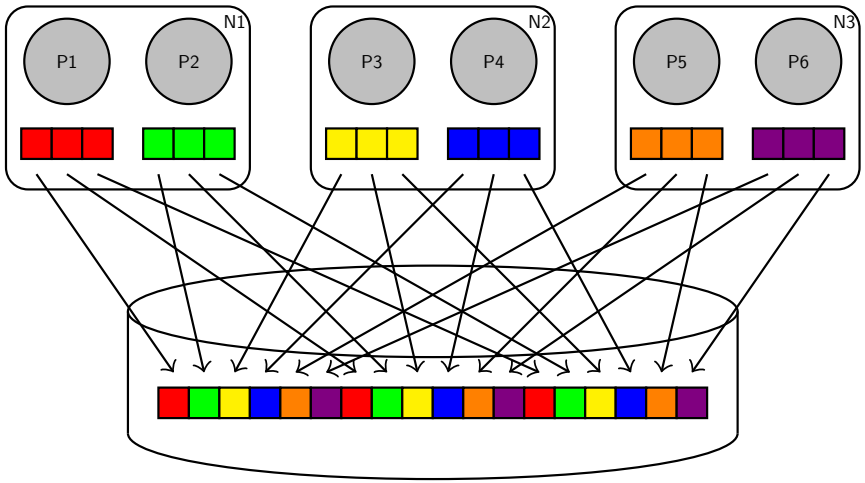


(N-N)

Checkpointing Modes
Shared-File



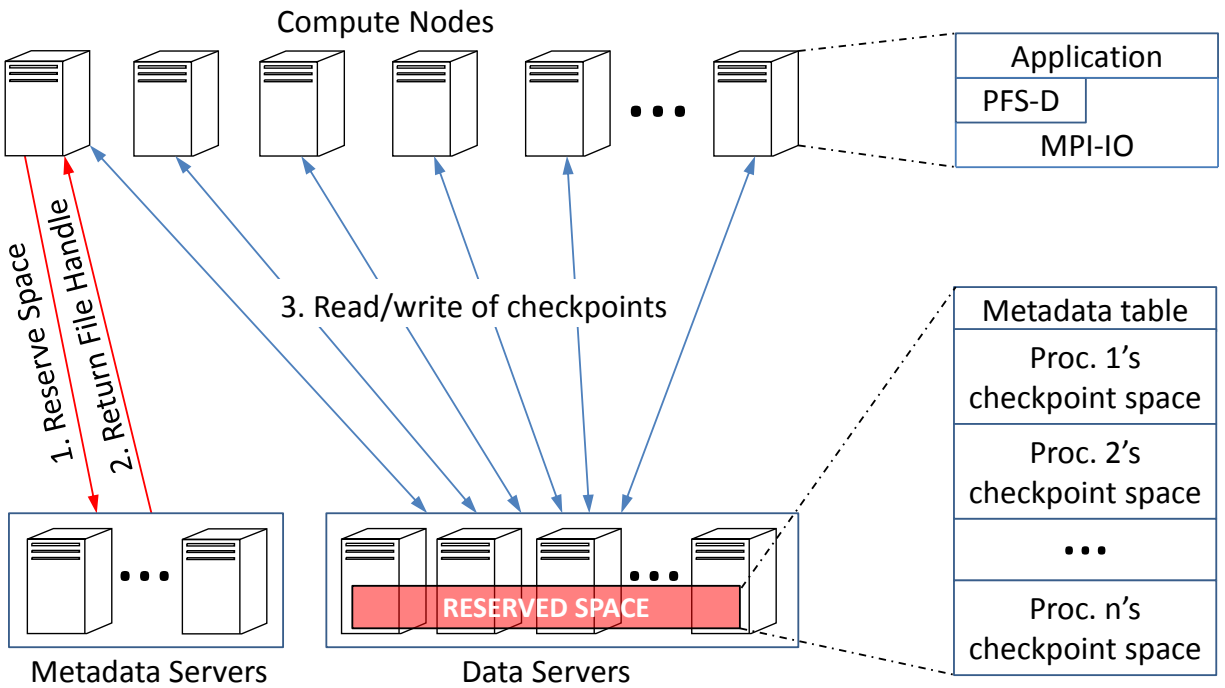
(N-1 Segmented)



(N-1 Strided)

- Shared-File (N-1) segmented
  - Processes write sequentially on shared-file's region
- Shared-File (N-1) strided
  - Processes write to different part of shared-file
- Metadata management overhead
  - Every process requests same metadata every time
  - File locking

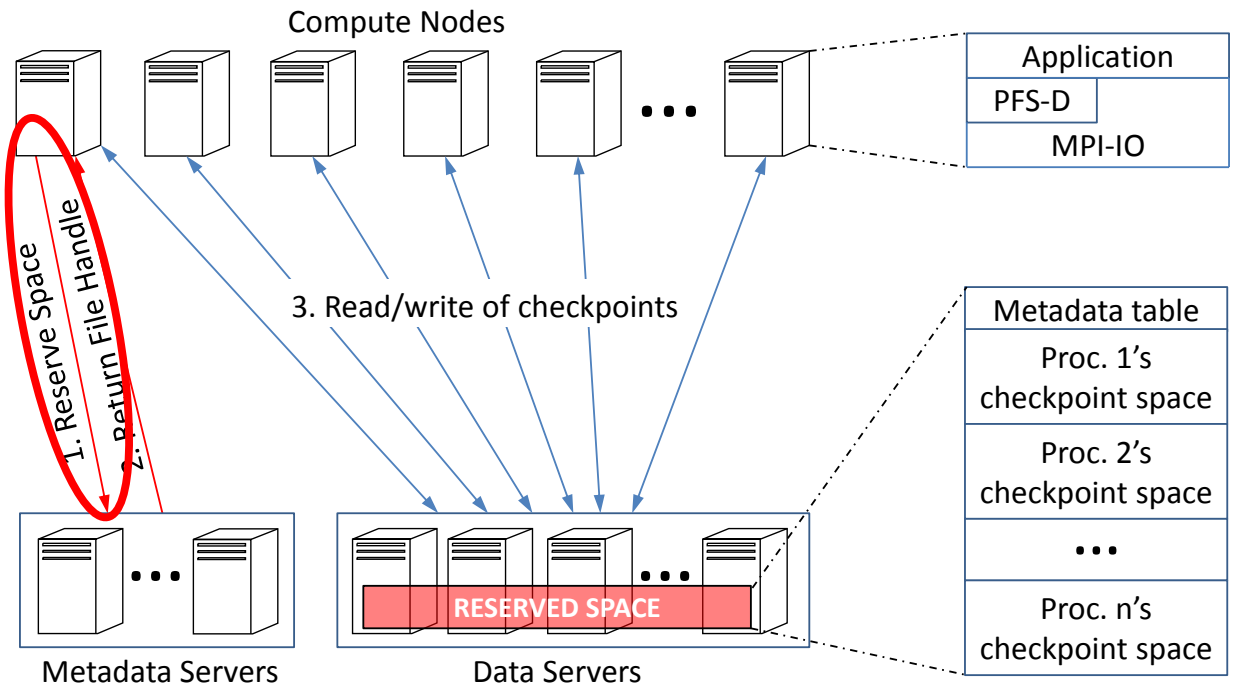
Approach - PFS-delegation



1. Create reserved space (only one time)
2. Receive metadata of reserved space (only one time)
3. Perform I/O directly to data servers

Read and write from/to checkpoints require to follow only step 3 after reserved space is created

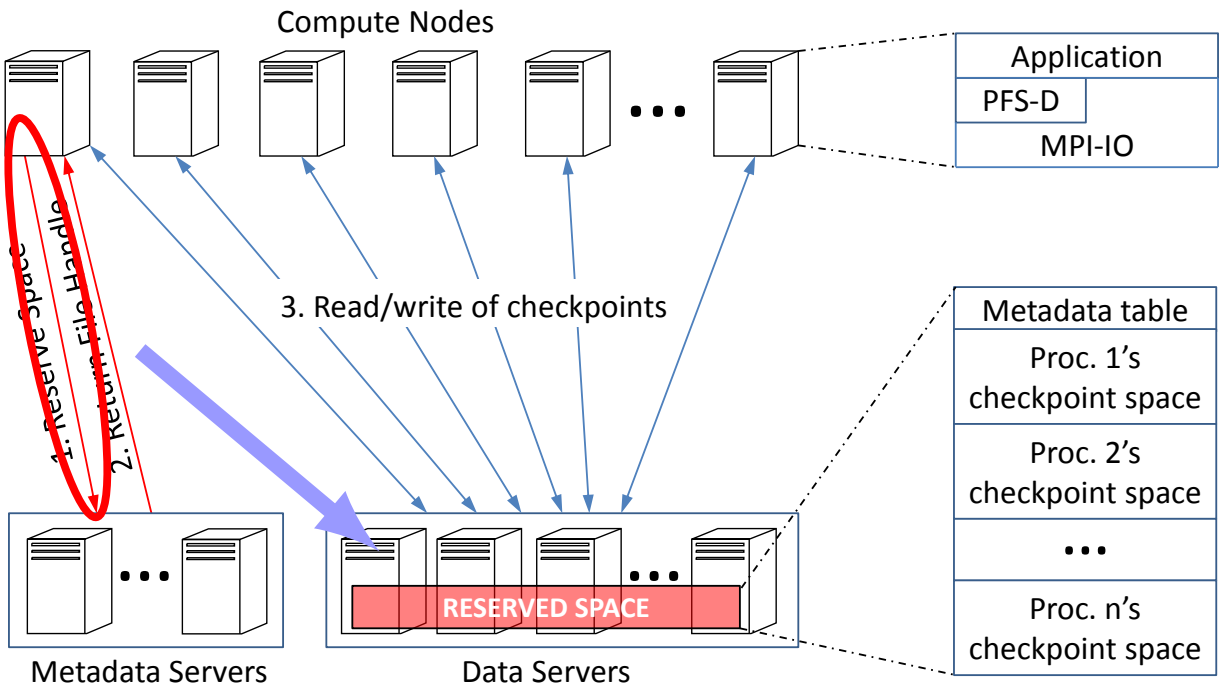
Approach - PFS-delegation



1. Create reserved space (only one time)
2. Receive metadata of reserved space (only one time)
3. Perform I/O directly to data servers

Read and write from/to checkpoints require to follow only step 3 after reserved space is created

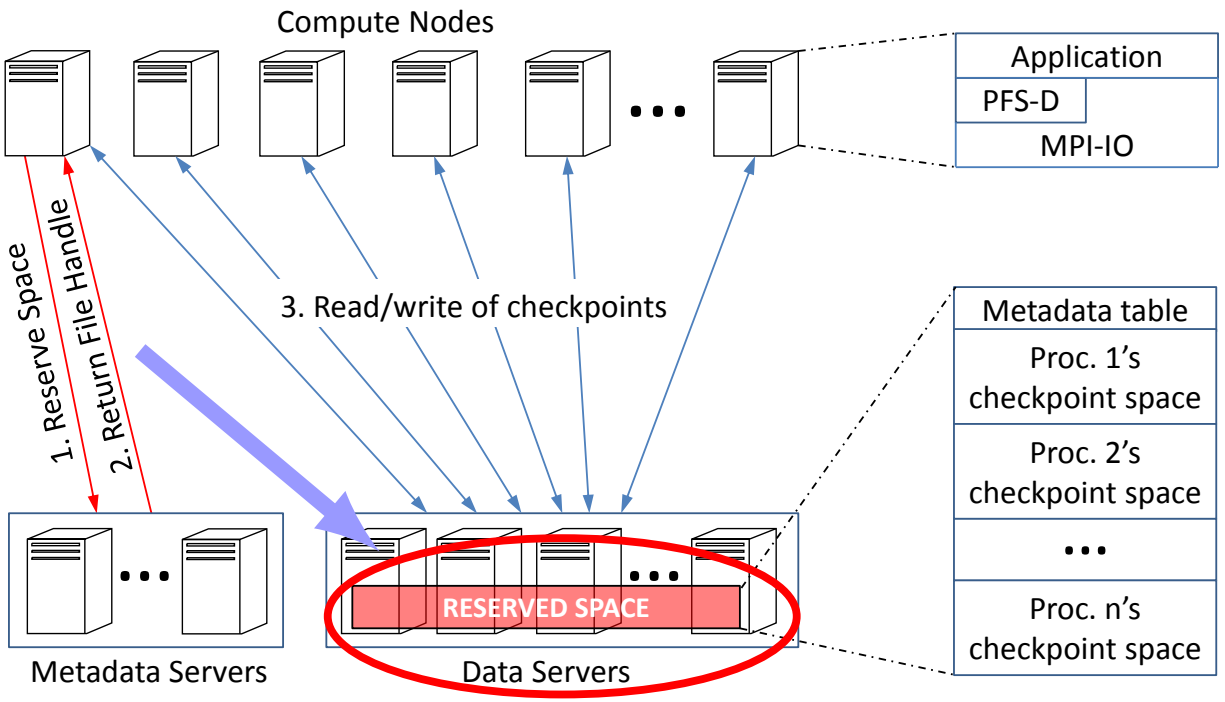
Approach - PFS-delegation



1. Create reserved space (only one time)
2. Receive metadata of reserved space (only one time)
3. Perform I/O directly to data servers

Read and write from/to checkpoints require to follow only step 3 after reserved space is created

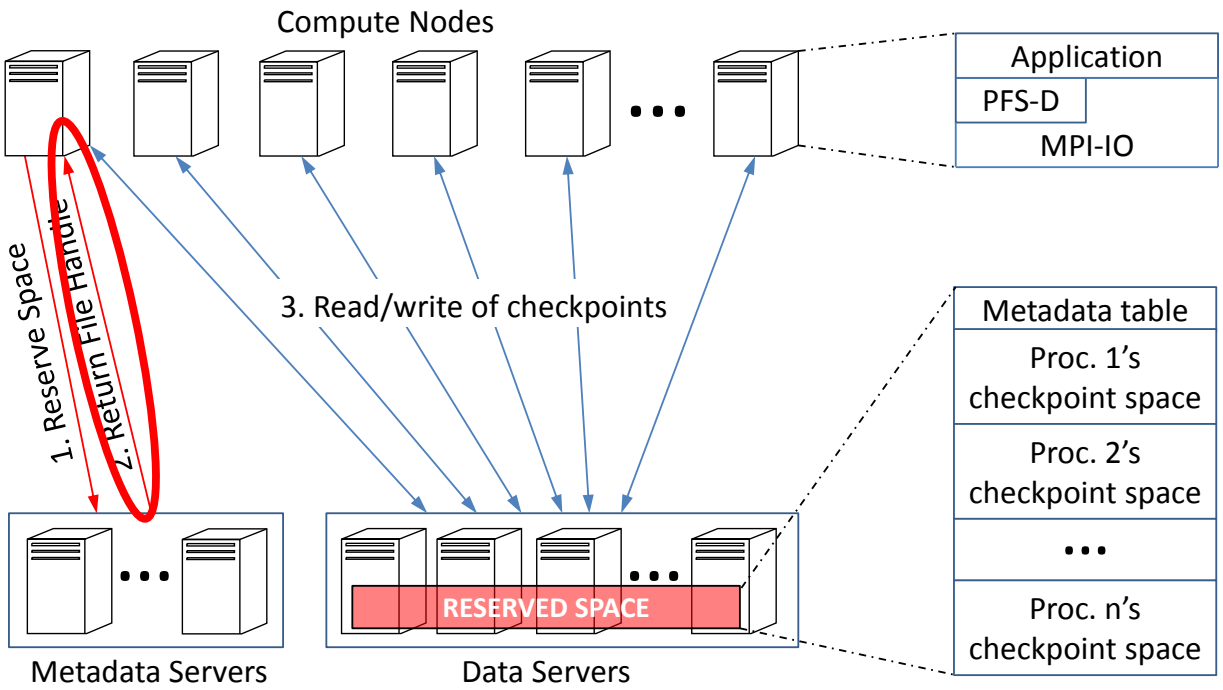
Approach - PFS-delegation



1. Create reserved space (only one time)
2. Receive metadata of reserved space (only one time)
3. Perform I/O directly to data servers

Read and write from/to checkpoints require to follow only step 3 after reserved space is created

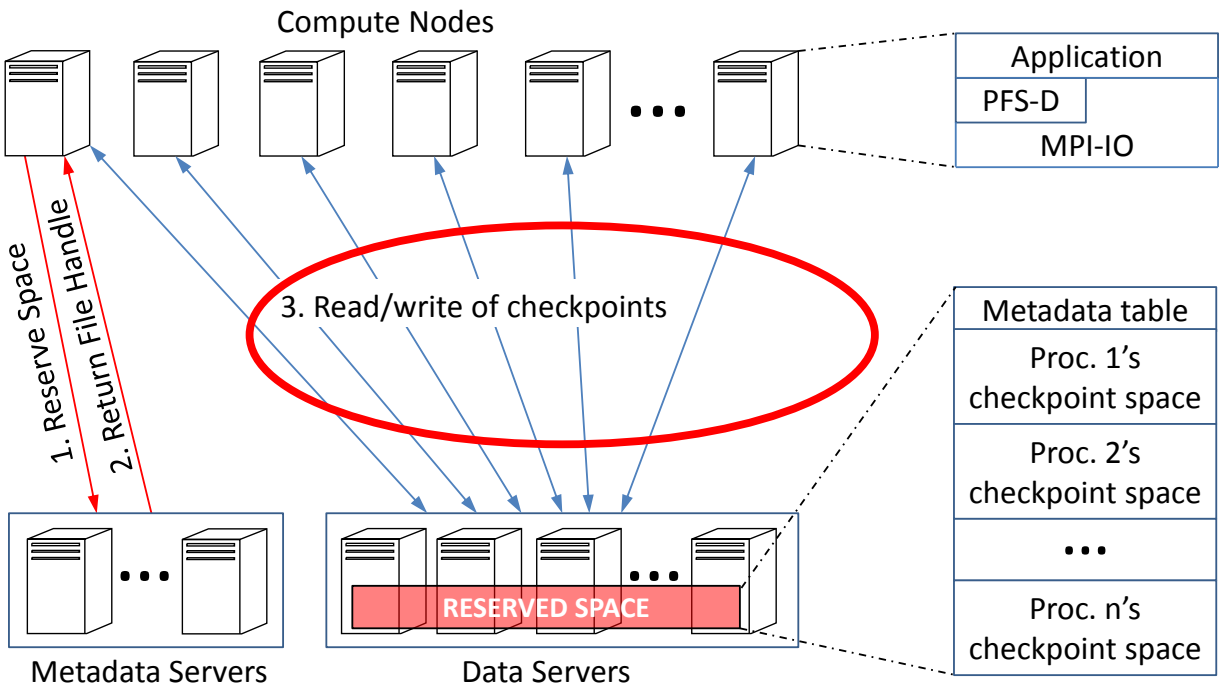
Approach - PFS-delegation



1. Create reserved space (only one time)
2. Receive metadata of reserved space (only one time)
3. Perform I/O directly to data servers

Read and write from/to checkpoints require to follow only step 3 after reserved space is created

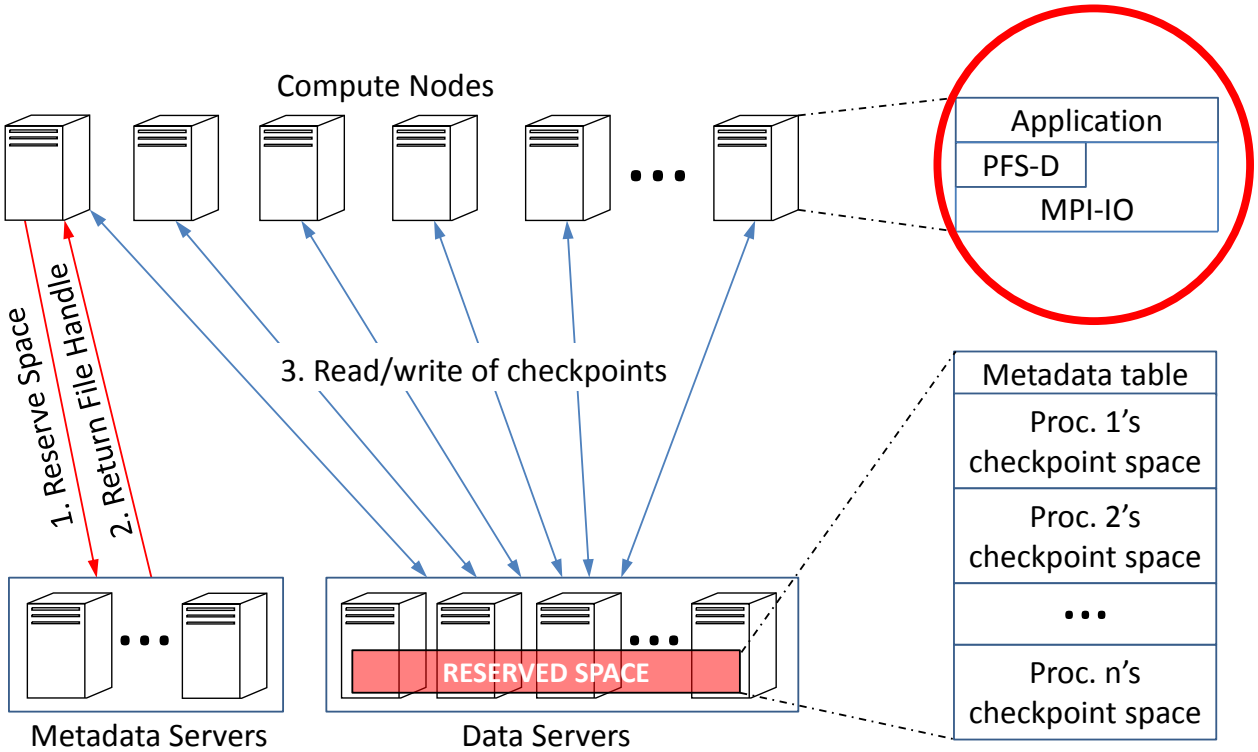
Approach - PFS-delegation



1. Create reserved space (only one time)
2. Receive metadata of reserved space (only one time)
3. Perform I/O directly to data servers

Read and write from/to checkpoints require to follow only step 3 after reserved space is created

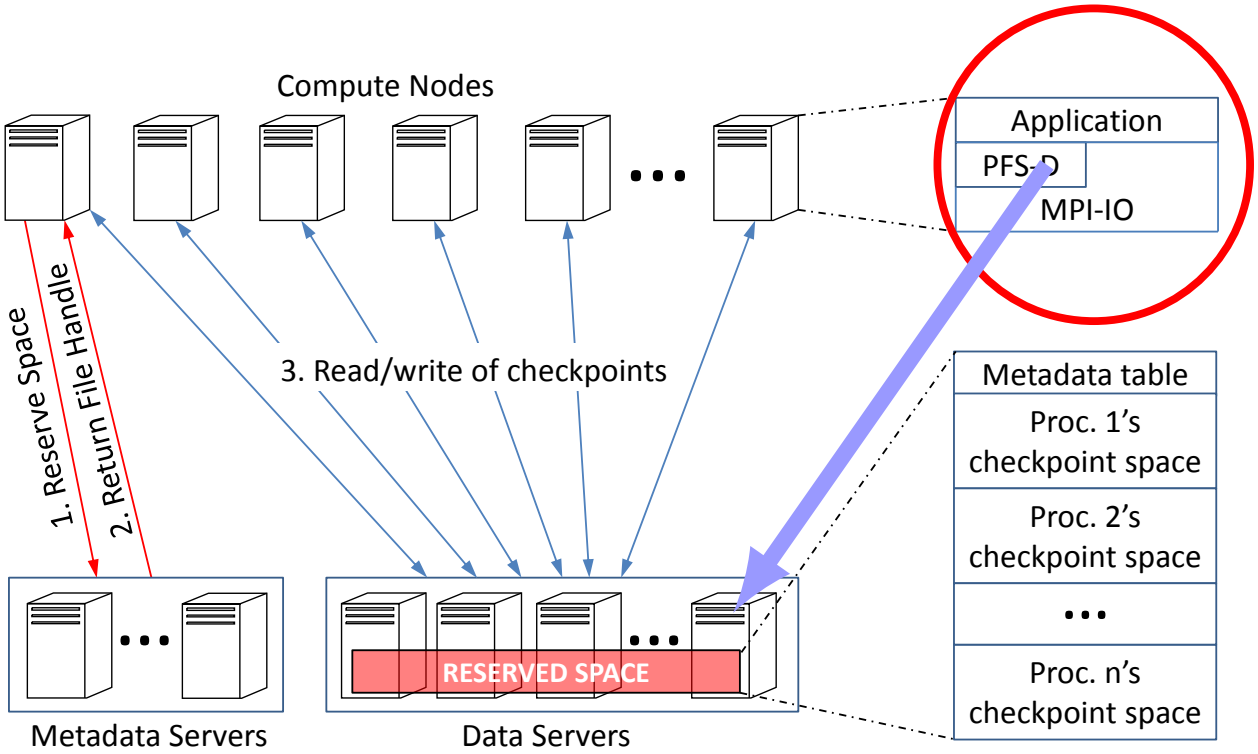
Approach - PFS-delegation



Application uses PFS-delegation interfaces

PFS-delegation uses MPI-IO API to communicate with servers

Approach - PFS-delegation

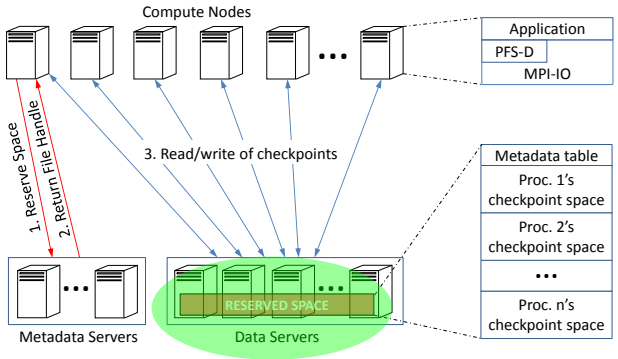


Application uses PFS-delegation interfaces

PFS-delegation uses MPI-IO API to communicate with servers

Reserving Delegated Storage Space

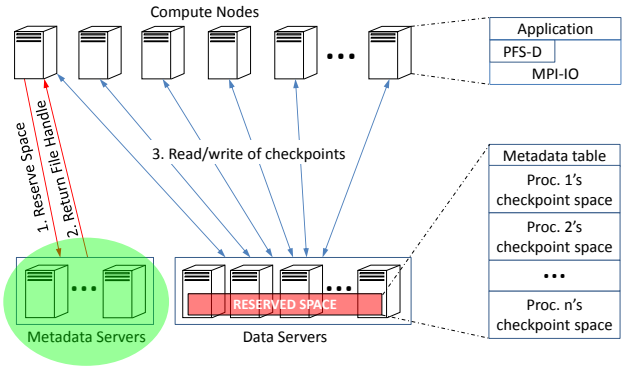
The reservation process is made by creating one large logical file across the PFS data servers



- To avoid initial overhead at reservation there are different techniques
  - Create a sparse file by writing the last byte of corresponding datafile (PVFS2)
  - Use fallocate (GPFS)
- This process is executed only once
- The size of reserved space should consider:
  - Single checkpointing size
  - Amount of checkpoints
  - Storage policy

Data Layout

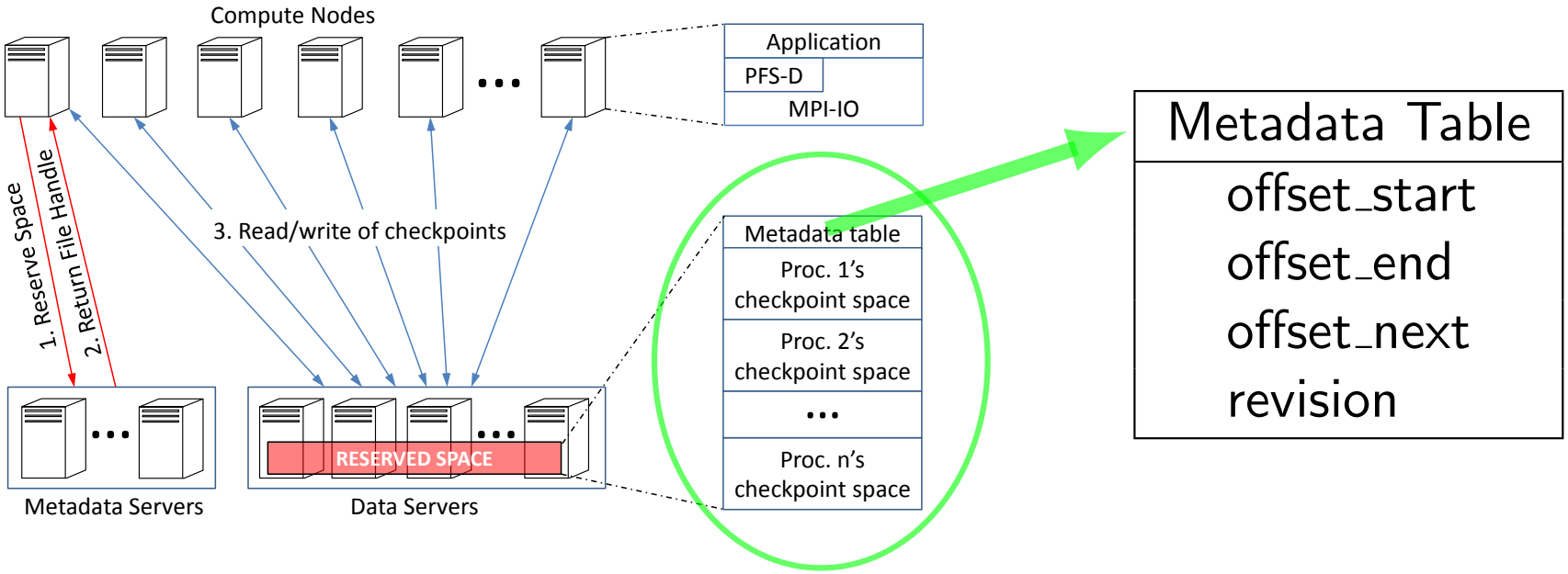
The layout is specified as a regular file layout using MPI-IO



- PFS-delegation uses the following hints for layout definition:
  - **striping_factor:** number of data server involved
  - **striping_unit:** stripe size
- PVFS2 implementation uses simple stripe and round robin distribution

```
MPI_info info;
MPI_Info_set(info, ‘‘striping_factor’’, ‘‘4’’);
MPI_Info_set(info, ‘‘striping_unit’’, ‘‘65536’’);
```

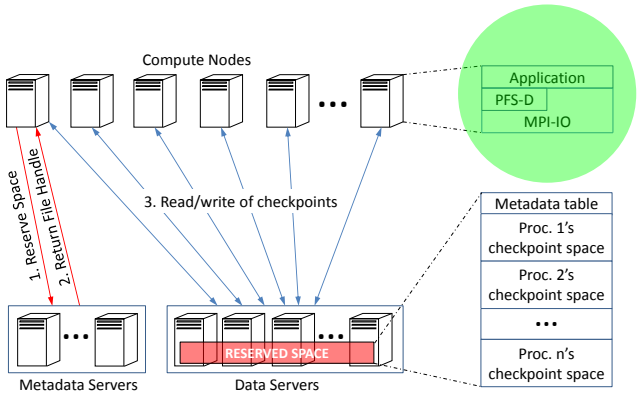
Reserved-Space Distribution



- **offset_start** and **offset_end**:
  - Specify limits of client's assigned region
- **offset_next**:
  - Specify next valid offset to write a checkpoint
- **revision**:
  - Checkpointing counter

Accessing Checkpoints in Delegated Space

PFS-delegation provides interfaces to applications to write/read checkpoints to the delegated space



Interface Name	Description
PFS_write_file	Perform writes of a checkpoint on the delegated space
PFS_read_file	Perform reads of last valid checkpoint from the delegated space
PFS_read_file_revision	Read a specific past checkpoint stored in the delegated space

Single Checkpoint Write Process

1. Read metadata table
2. Get the offset "offset_next" (available space)
3. Call MPI-IO functions to do write
4. Update metadata table with new offsets
5. Increase revision number

- Only one process (rank 0) performs lookup and update to metadata table
- In case of N-N and N-1 modes many processes update metadata info

Single Checkpointing Read Process

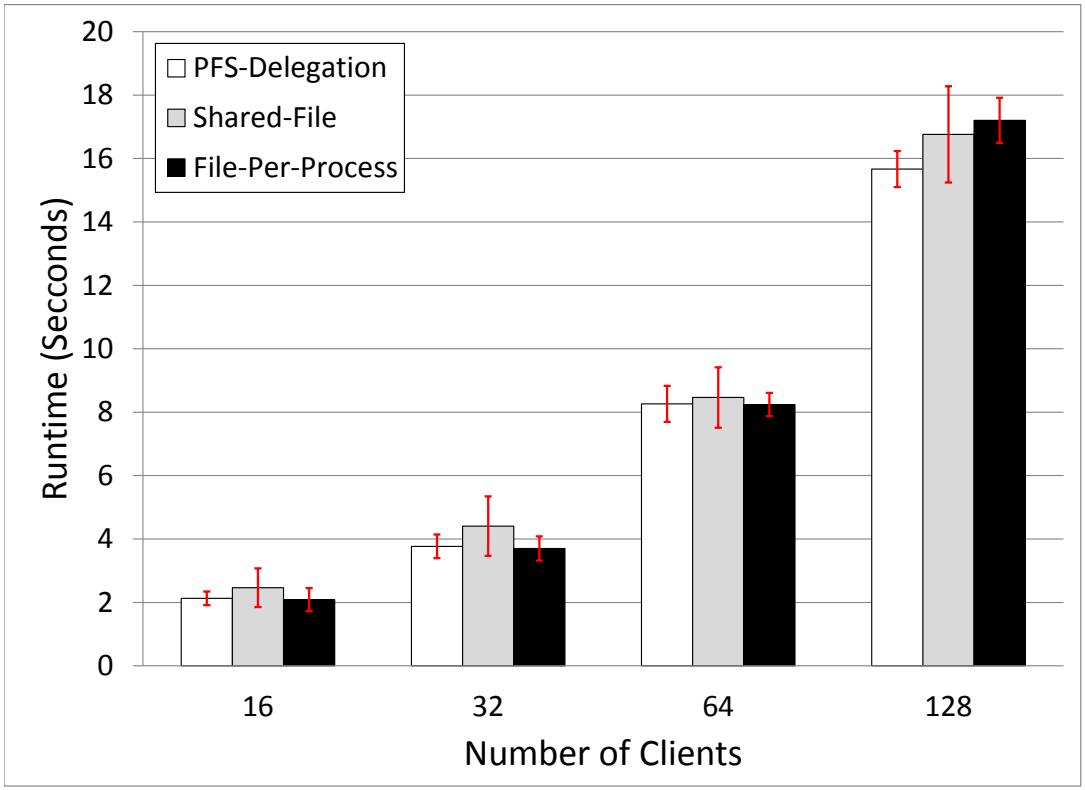
1. Read metadata table
2. Get corresponding offset where the data is located
3. Call MPI-IO functions to perform read in parallel

- Only one process (rank 0) performs lookup at metadata table
- In case of N-N and N-1 modes many processes update metadata info

- Evaluation was performed in our cluster:
  - Eleven DELL cluster nodes
  - 2 six-core 2.4GHz Opteron
  - 32GB RAM - 500GB SAS Disk
  - OS: Ubuntu 8.04 Kernel: 2.6.24-16-server
- Benchmark IOR2

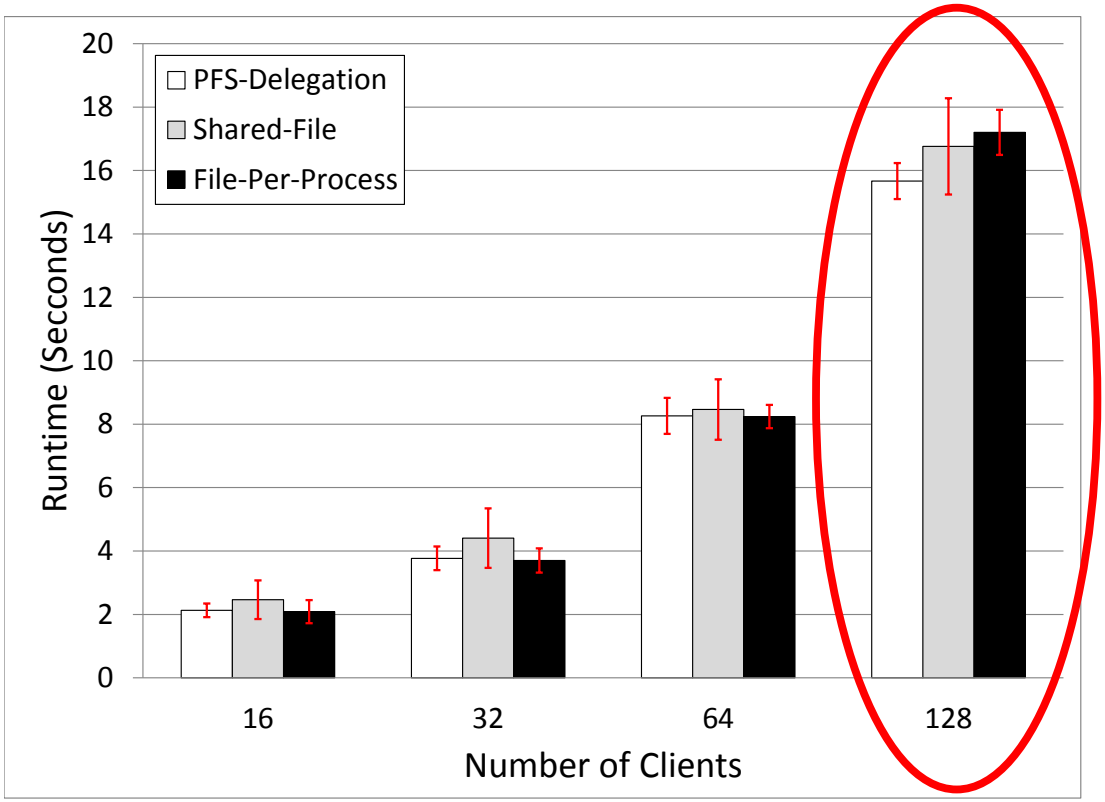
	Distributed Metadata Server	Centralized Metadata Server
Node 1 to Node 4	4 Meta Servers 4 Data Servers	4 Data Servers
Node 5		1 Meta Server
Node 6 to Node 11	16 to 128 Processes	16 to 128 Processes

Centralized Metadata Server
Checkpointing Time



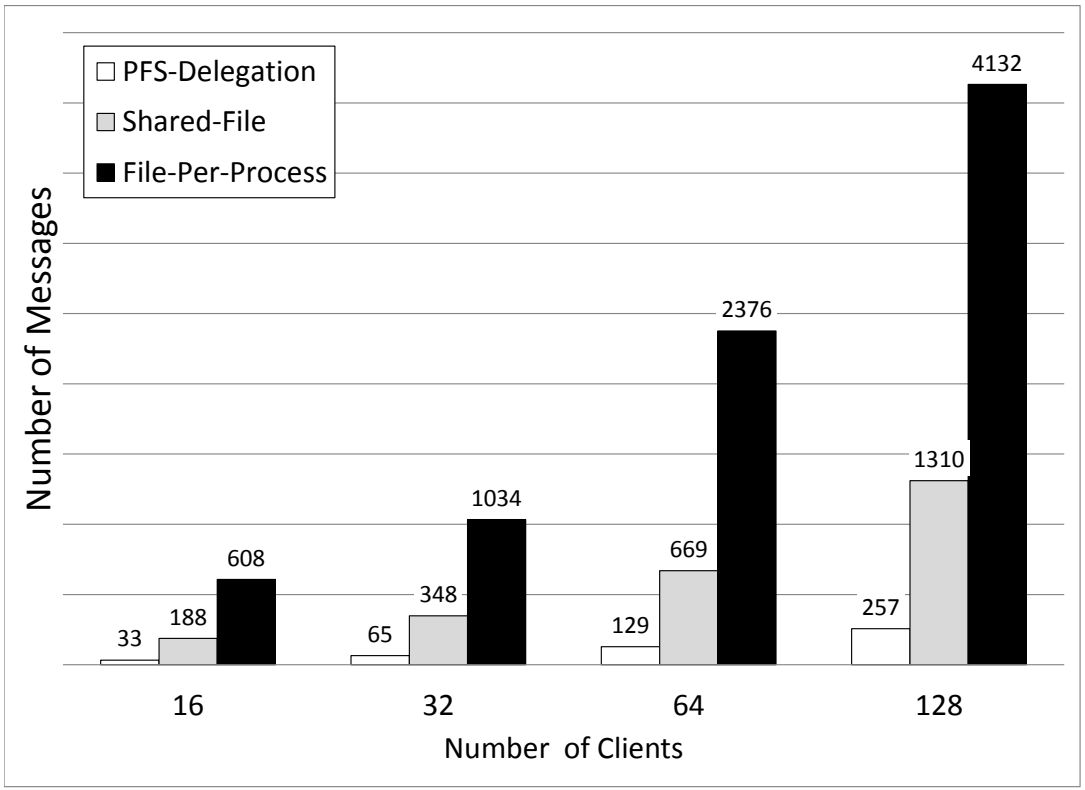
- Performance is similar with less than 64 clients
- With 128 clients PFS-delegation is:
  - 7% faster than "shared-file"
  - 10% faster than "file-per-process"

Centralized Metadata Server
Checkpointing Time



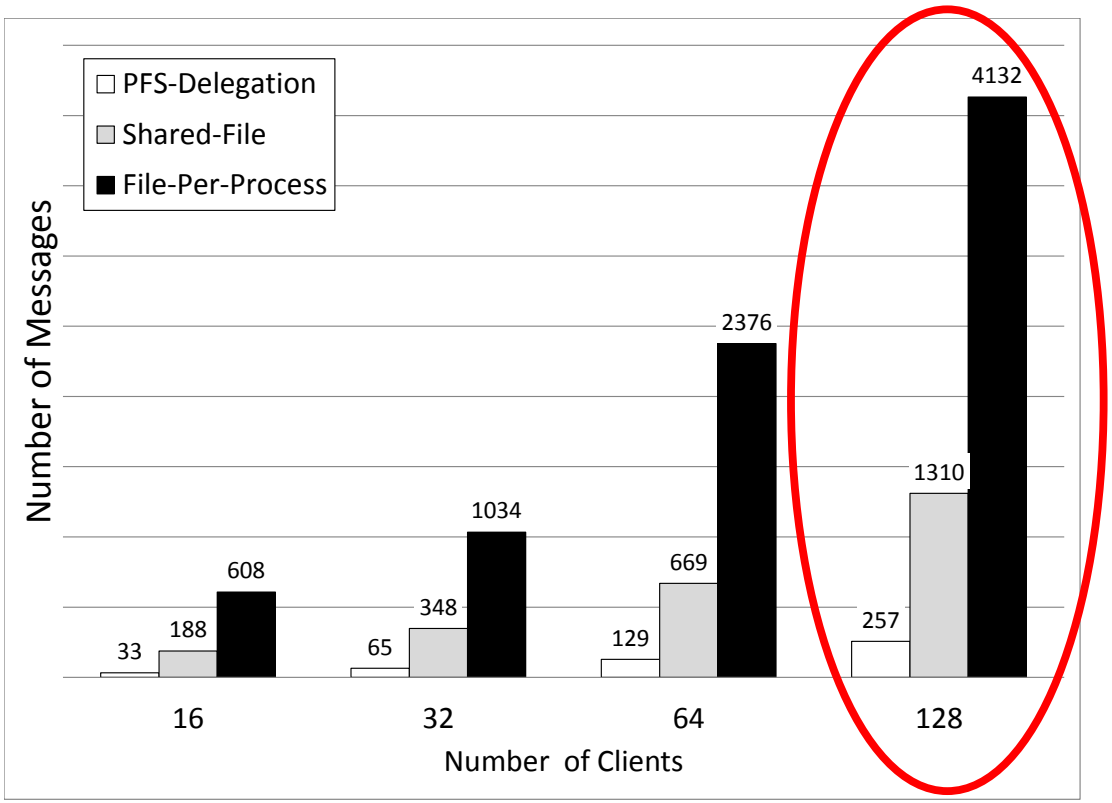
- Performance is similar with less than 64 clients
- With 128 clients PFS-delegation is:
  - 7% faster than "shared-file"
  - 10% faster than "file-per-process"

Centralized Metadata Server
Total number of Metadata Operations



- PFS-delegation metadata operations reduced to:
  - 7% of "file-per-process"
  - 20% of "shared file"
- With 128 processes the metadata operations are reduced by:
  - 1053 compared to "shared file"
  - 3875 compared to "file-per-process"

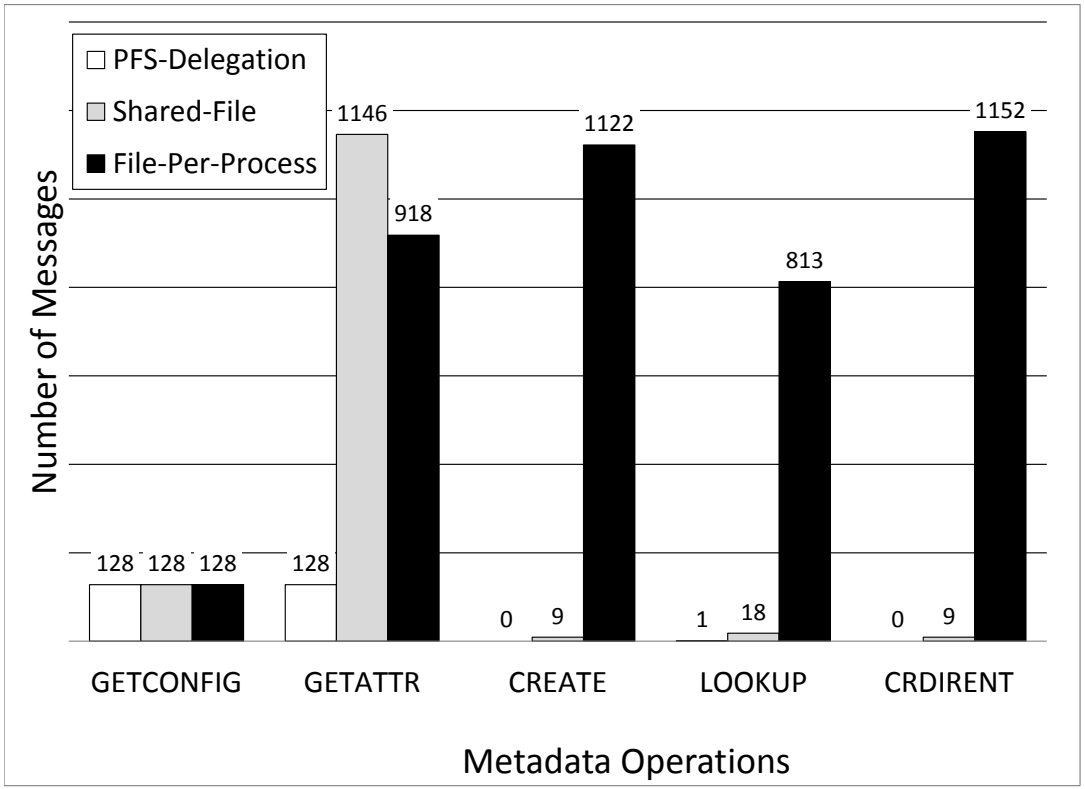
Centralized Metadata Server
Total number of Metadata Operations



- Metadata operations reduced to:
  - "shared-file" is 30% of "file-per-process"
  - "PFS-delegation" is 20% of "shared file"
- With 128 processes the metadata operations are reduced by:
  - 1053 compared to "shared file"
  - 3875 compared to "file-per-process"

Centralized Metadata Server

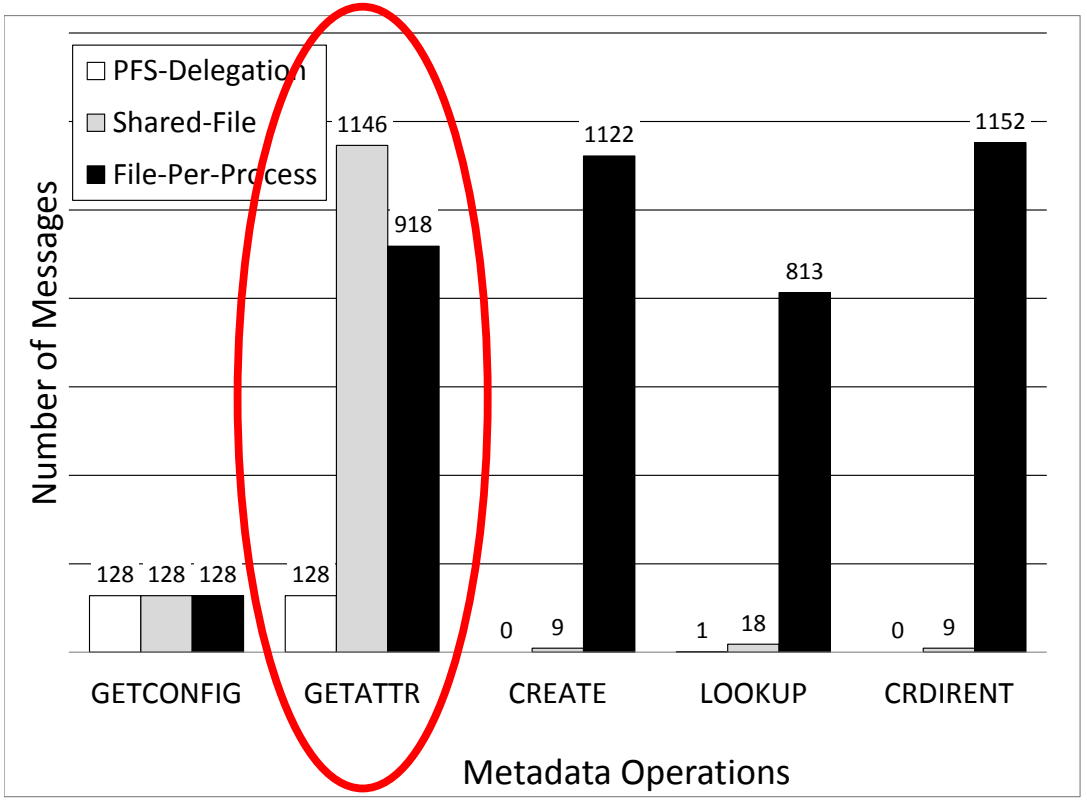
Different metadata operations with 128 processes



- PFS-delegation's "GETATTR" is less than the other two methods
  - Triggered by: create, read, and write

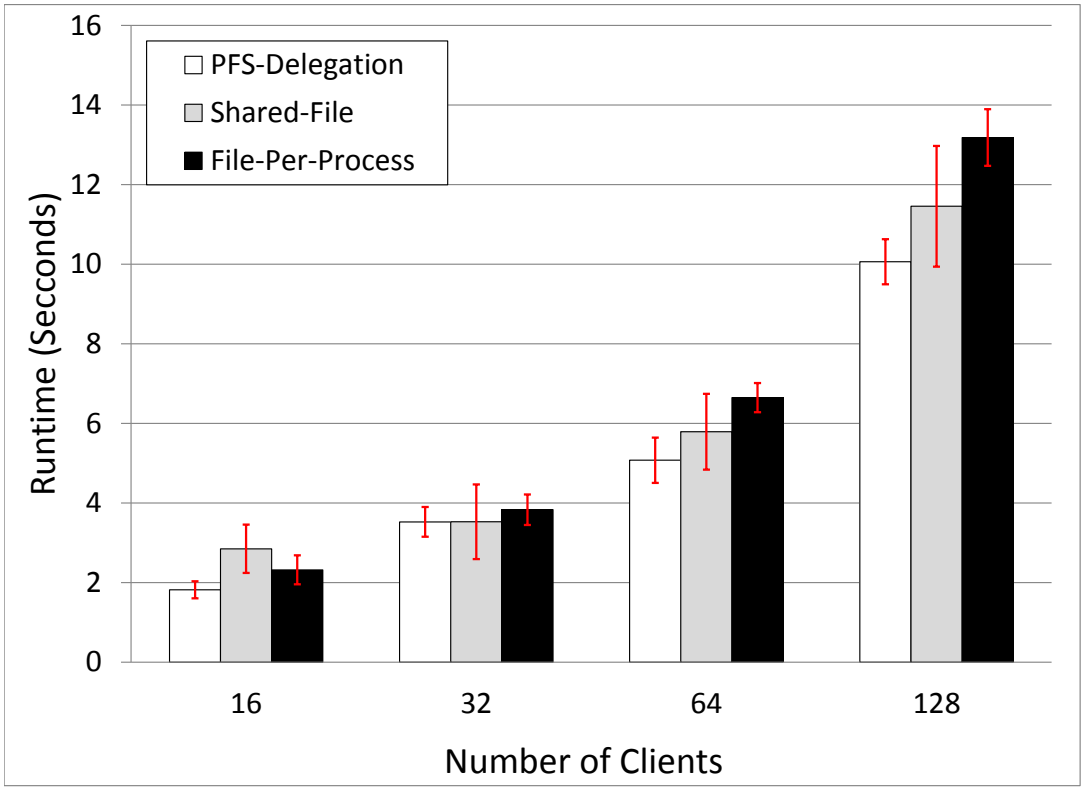
Centralized Metadata Server

Different metadata operations with 128 processes

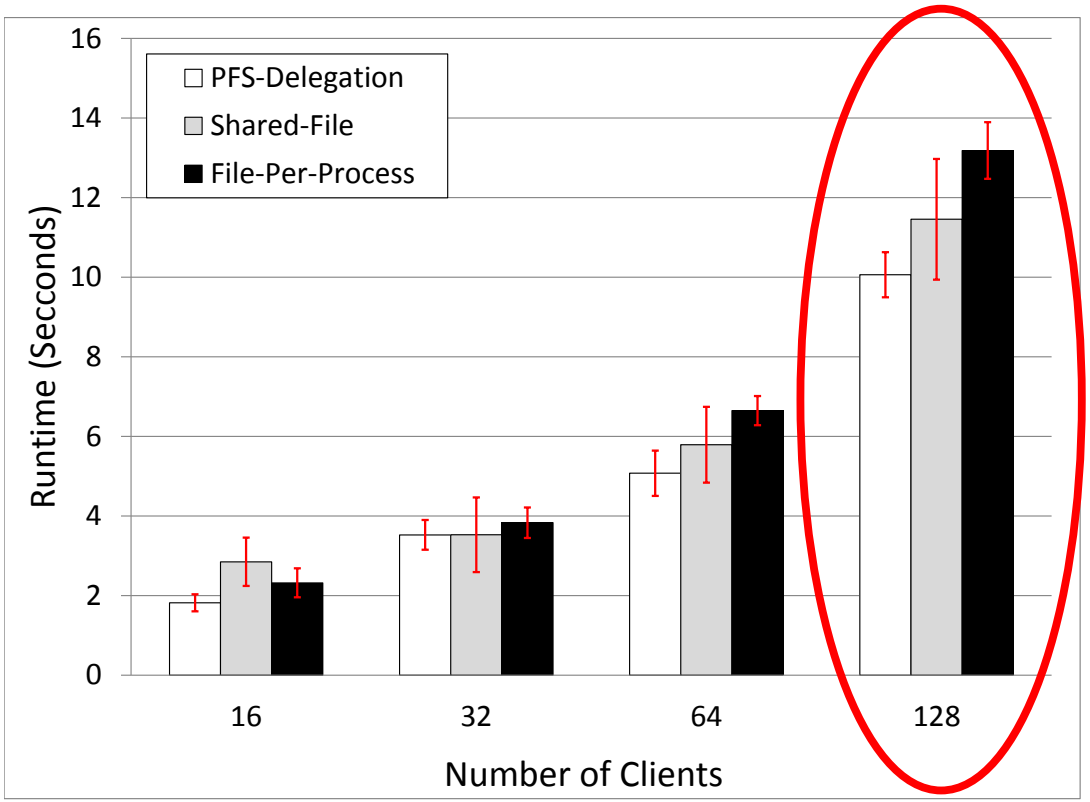


- PFS-delegation's "GETATTR" is less than the other two methods
  - Triggered by: create, read, and write

- Performance is similar with less than 32 clients
- With 128 clients PFS-delegation is:
  - 22% faster than "shared-file"
  - 31% faster than "file-per-process"

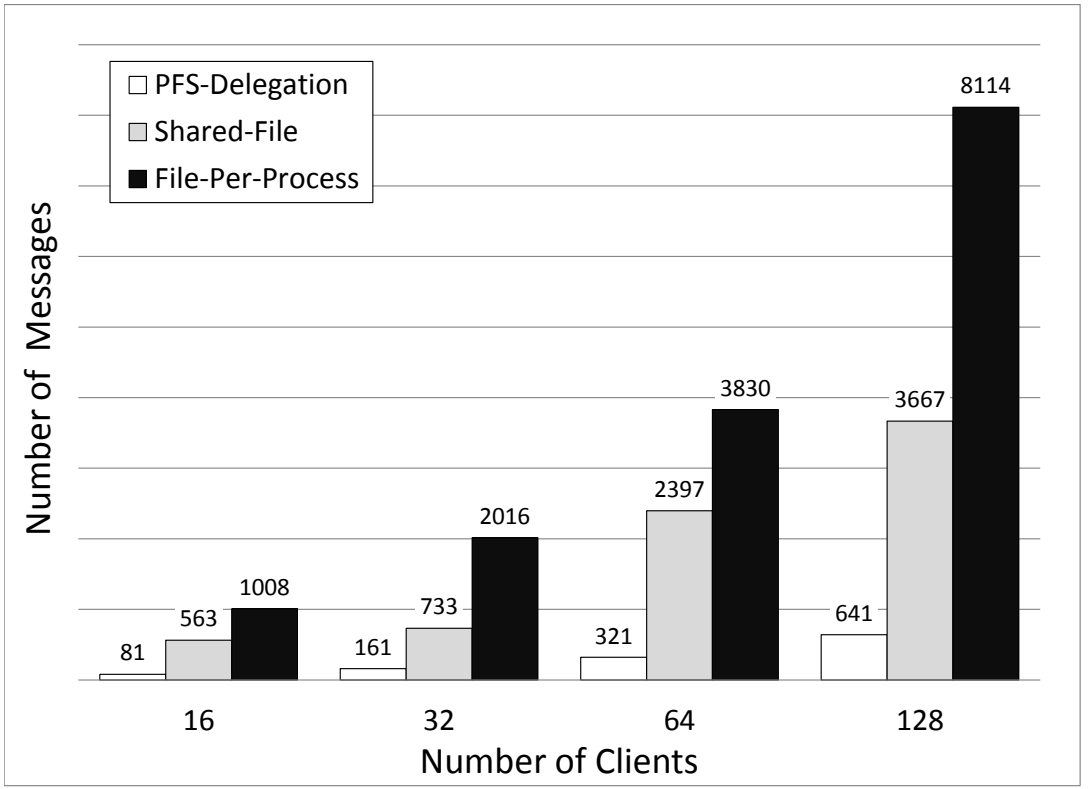


- Performance is similar with less than 32 clients
- With 128 clients PFS-delegation is:
  - 22% faster than "shared-file"
  - 31% faster than "file-per-process"



Distributed Metadata Server
Total number of Metadata Operations

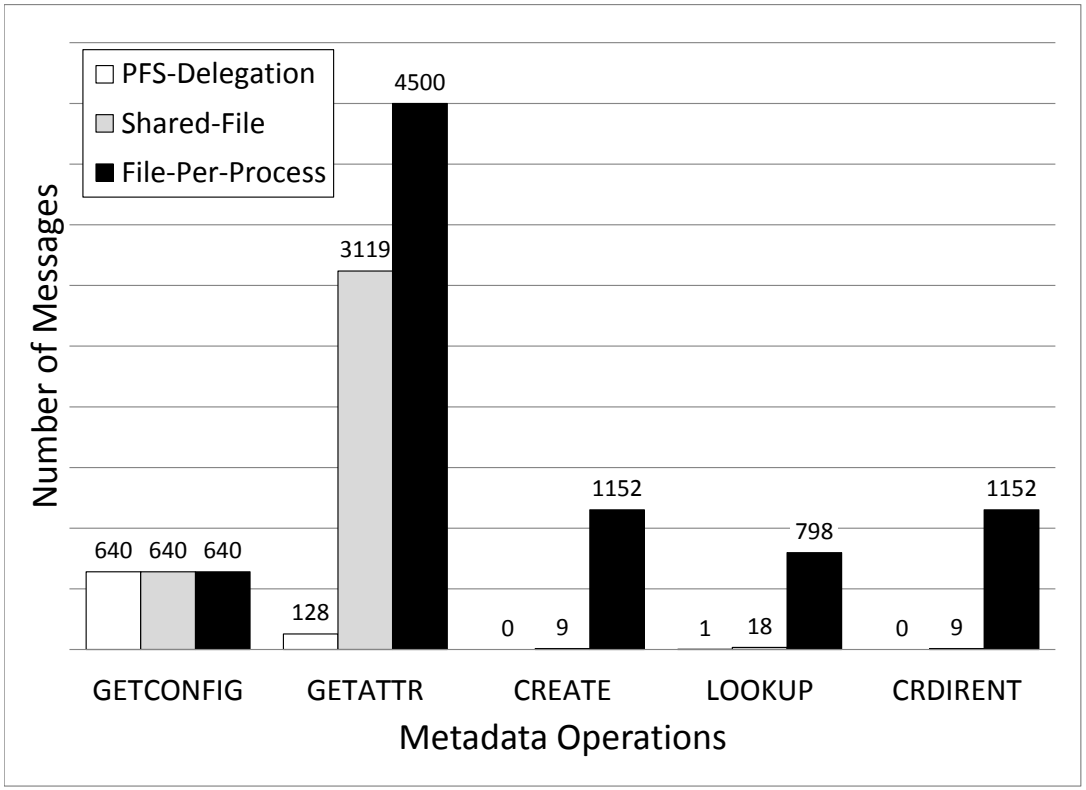
- More metadata operations than centralized metadata server
- Metadata operations reduced to:
  - 20% of "shared file"
  - 10% of "file-per-process"



Distributed Metadata Server

Different metadata operations with 128 processes

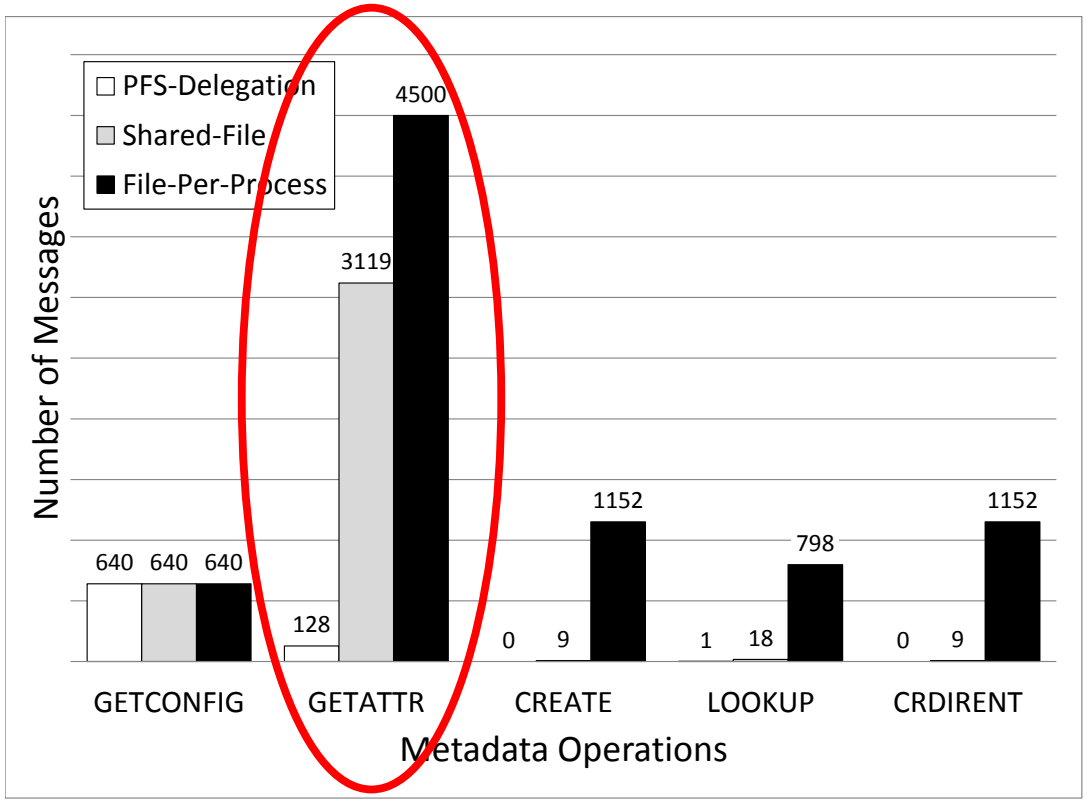
- PFS-delegation's "GETATTR" is much less than shared-file/file-per-process



Distributed Metadata Server

Different metadata operations with 128 processes

- PFS-delegation's "GETATTR" is much less than shared-file/file-per-process



Related Work

- PLFS - Parallel Log structure File System
  - Map the access pattern from N-1 to N-N
  - Create interposition layer between application and PFS
  - Implements access transparently by providing *ad_plfs* MPI-IO driver
- GFS - Google File System
  - Handle large workloads
  - Perform better with appending-only writes
- LWFS - Light-Weight File System
  - No traditional PFS services
  - Provide secure access and high-level services

Conclusions

- PFS-delegation is a checkpointing technique that reduces the overhead at metadata management
  - Require no modifications on PFS
  - Provide simple interfaces to applications

- A prototype on PVFS2 was implemented with good results compared to shared-file and file-per-process
  - 7% and 10% speedup using centralized metadata server
  - 22% and 31% speedup using distributed metadata server

Future Work

- Implement PFS-delegation MPI-IO driver to provide full transparency to application
- Integrate PFS-delegation capabilities to use netCDF/HDF5 to structure the reserved space
- Scale up the number of clients/server for future experiments

Acknowledgment

This research is sponsored by National Science Foundation under grant **CCF-0938045** and Department of Homeland Security under grant **2010-ST-062-00039**

Questions?

WEB: http://visa.cis.fiu.edu

darte003@fiu.edu ming@cs.fiu.edu