



**Argonne**  
NATIONAL  
LABORATORY

*... for a brighter future*



U.S. Department  
of Energy

UChicago ►  
Argonne<sub>LLC</sub>



A U.S. Department of Energy laboratory  
managed by UChicago Argonne, LLC

# Scalable I/O Forwarding Framework for HPC Systems

**Nawab Ali: The Ohio State University**

**Philip Carns, Kamil Iskra, Dries Kimpe, Samuel  
Lang, Robert Latham, Robert Ross:  
Argonne National Laboratory**

**Lee Ward: Sandia National Laboratories**

**P. Sadayappan: The Ohio State University**



# Compute and storage imbalance

- Leadership class computational scale:
  - 100,000+ processes
  - Advances in multi-core architectures and power efficiency
  - Minimal compute node operating system
- Leadership class storage scale:
  - 100+ servers
  - Commercial storage hardware
  - Cluster file systems

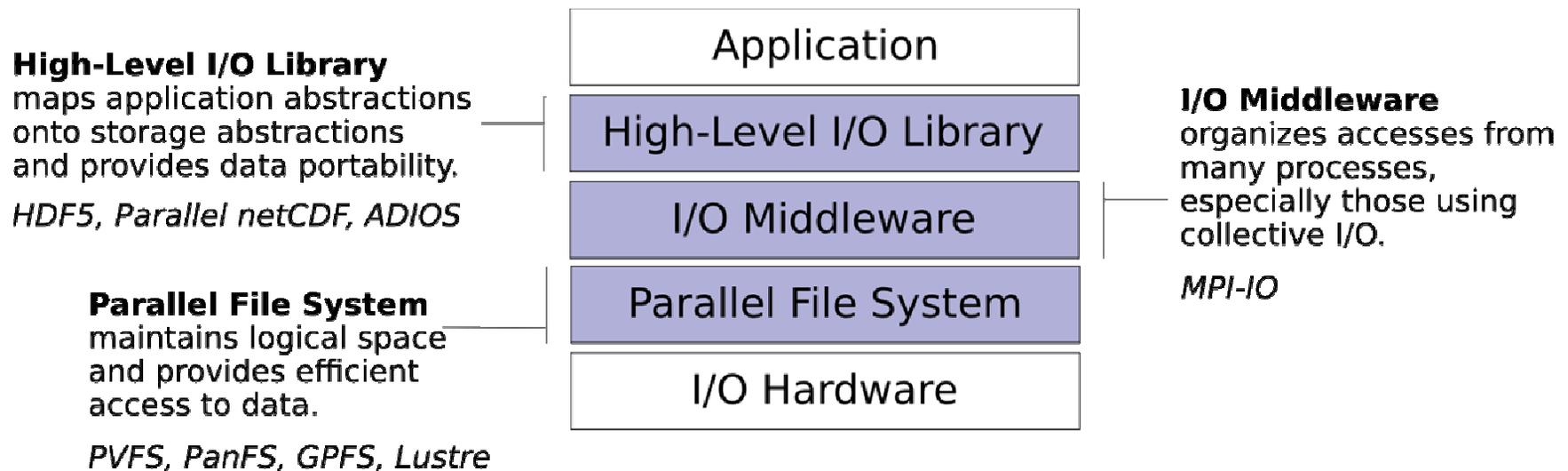
Current leadership-class machines supply only **1 GB/s of storage throughput for every 10 TF of compute performance**. This gap has grown by a factor of 10 in recent years.

Bridging the imbalance between compute and storage is critical a critical concern for the I/O software environment on leadership machines.



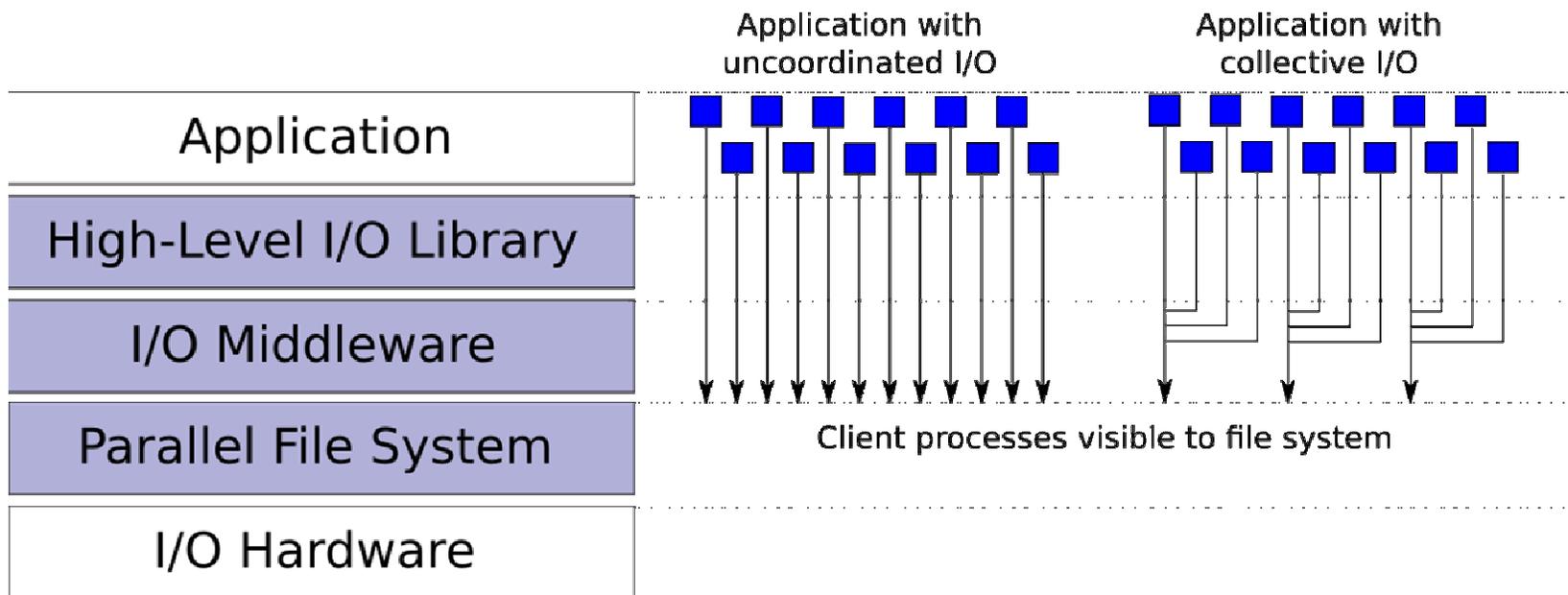
# I/O software architecture

The file system is not the only software component of an HPC storage system. HPC systems rely on a combination of software to meet the needs of scientific applications.



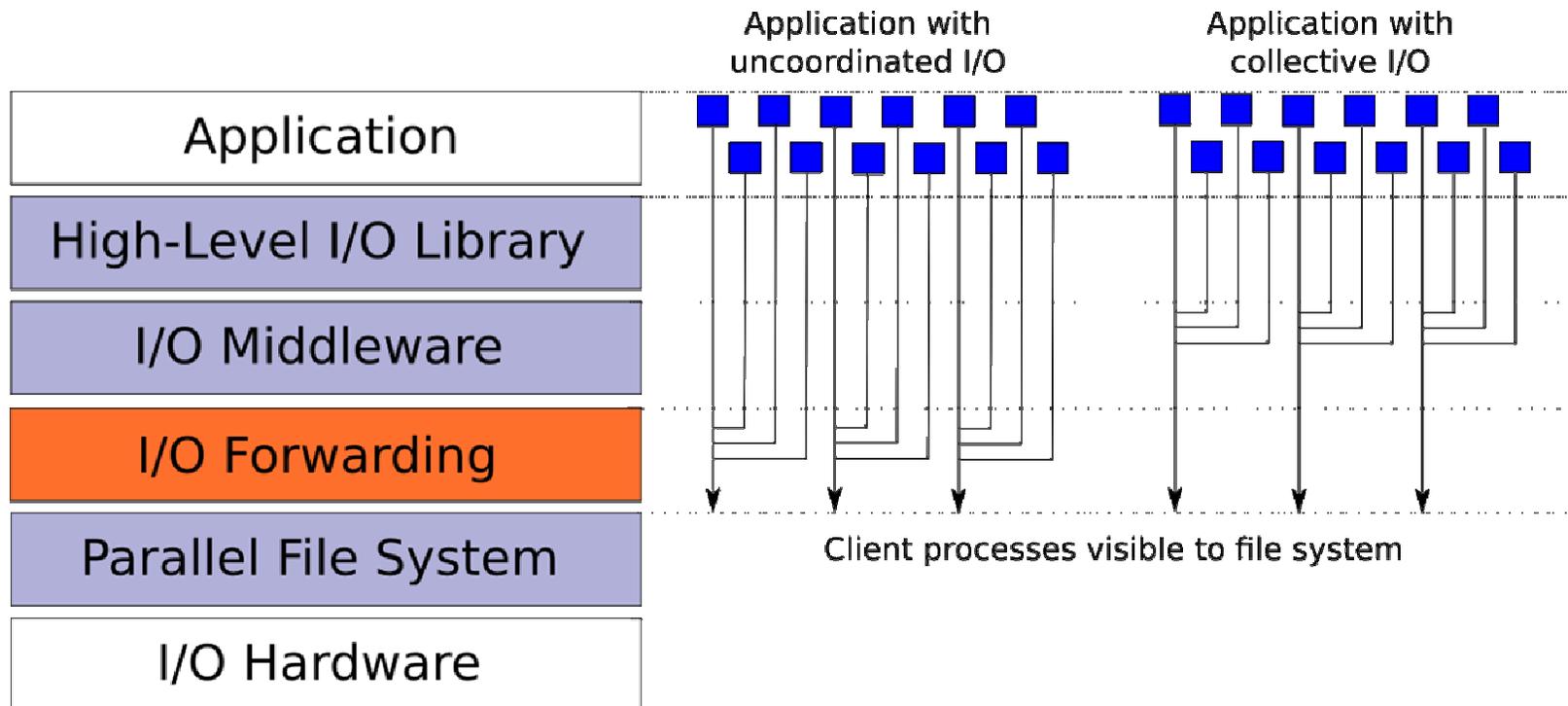
# Aggregation

- 100,000+ concurrent clients present a challenge to parallel file systems.
- Aggregation improves performance by transforming access patterns into workloads that are better tuned for file system characteristics.
- I/O middleware can aggregate access, but only for a subset of applications using specific collective operations.



# Bridging the scalability gap

- I/O Forwarding is an additional I/O software layer for leadership class machines that bridges the gap between application process and file systems. It aggregates uncoordinated file access, thereby reducing the number of clients seen by the file system for all applications.

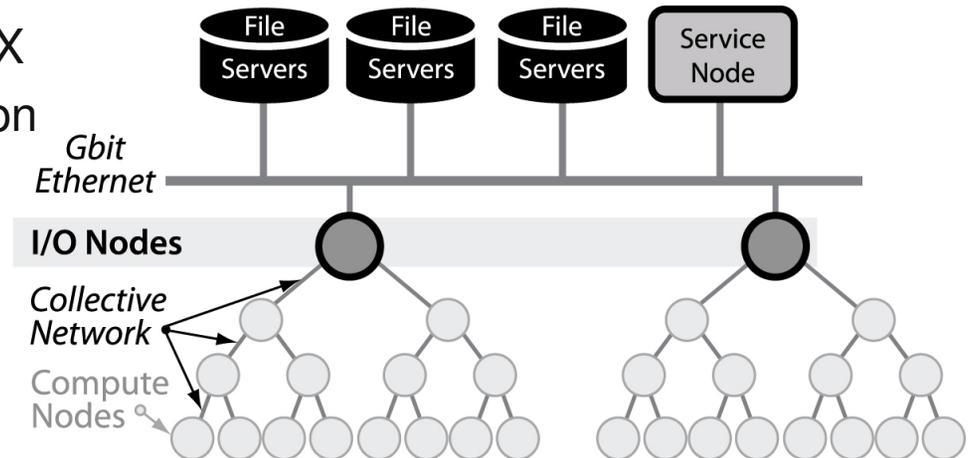


# Previous work in I/O forwarding

- I/O forwarding was first introduced on the Cplant machine at Sandia
- Now a standard component of systems such as the IBM Blue Gene series
- The Blue Gene forwards all I/O operations to dedicated I/O nodes
  - Reduces OS noise
  - Compute OS can be stripped down to minimal functionality
  - Aggregate access to file system

Drawbacks of current implementations:

- Lack of API support beyond POSIX
- Machine-dependent implementation
- Limited extensibility



# I/O Forwarding Scalability Layer (IOFSL)

Design, build, and distribute a scalable, unified high-end computing I/O forwarding software layer that would be adopted and supported by DOE Office of Science and NNSA.

- Provide function shipping at the file system interface level
- Offload file system functions from simple or full OS client processes to a variety of targets
- Reduce the number of file system operations/clients that are visible to the parallel file system
- Support multiple parallel file systems and networks
- Integrate with MPI-IO and any hardware features designed to support efficient parallel I/O

This presentation presents work from the first IOFSL prototype.



# ZOIDFS Protocol

- The I/O forwarding framework uses ZOIDFS rather than POSIX to express I/O operations
  - NFSv3-like protocol
  - Opaque, 32-byte handles used to reference files
    - *Portable across nodes*
    - *Obtained via lookup() or create()*
    - *No close() operation*
    - *No stateful file descriptor*
  - Flexible read() and write() operations
    - *Vectors of memory buffers and file regions*
    - *Explicit offsets*
- Minimizes state to improve scalability
- Reduces the number of I/O operations
- Enables middleware optimizations
  - Example: lookup handle on one node and broadcast to application



# ZOIFFS Protocol Examples

```
int zoidfs_lookup(const zoidfs_handle_t *parent_handle,  
                 const char *component_name,  
                 const char *full_path,  
                 zoidfs_handle_t *handle);
```

```
Int zoidfs_read(const zoidfs_handle_t *handle, int mem_count,  
               void *mem_starts[],  
               const size_t mem_sizes[],  
               int file_count,  
               const uint64_t file_starts[],  
               uint64_t file_sizes[]);
```

- Note that file and memory buffers are specified independently
- Final protocol will also include credentials and hints

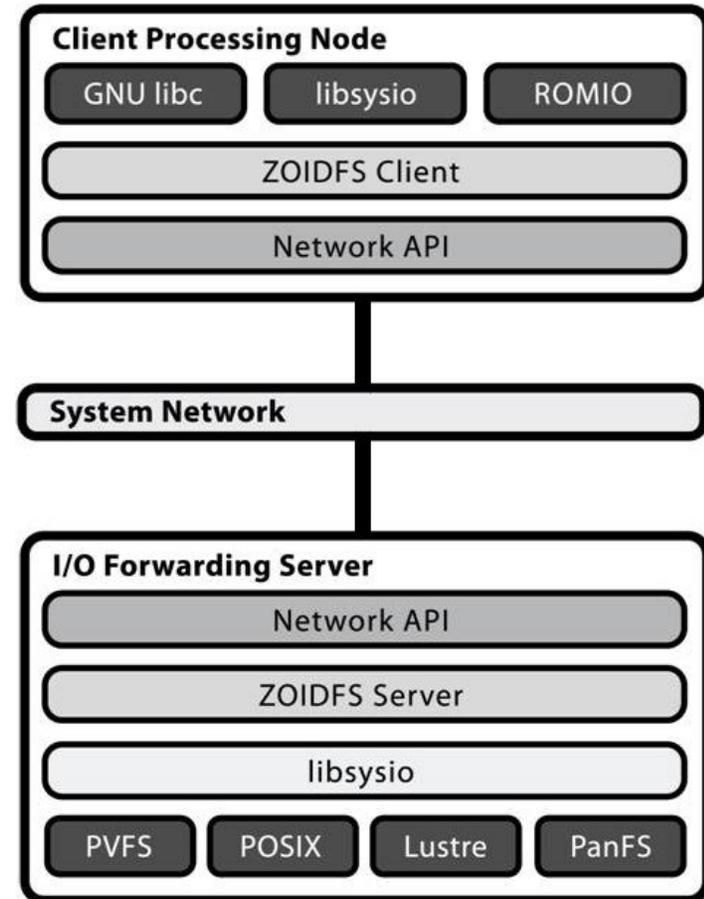
# I/O Forwarding Architecture

## Client:

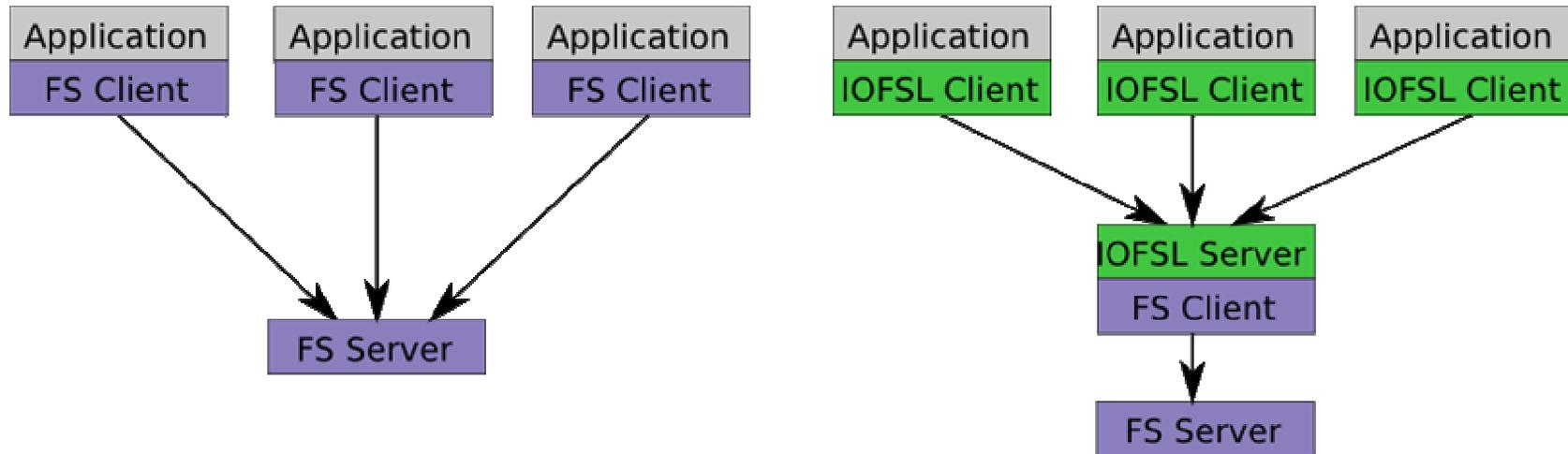
- POSIX, MPI-IO, or SYSIO interface
  - MPI-IO via ROMIO abstract device
  - POSIX via wrappers or FUSE
- Encodes requests
- Transmits to server via network API
  - BMI originally developed for PVFS

## Server:

- Decodes request
- Translates to back end file systems via SYSIO or native drivers
- BMI provides network API abstraction
  - Originally developed for PVFS file system
  - Supports TCP, Myrinet, Infiniband, and Portals networks



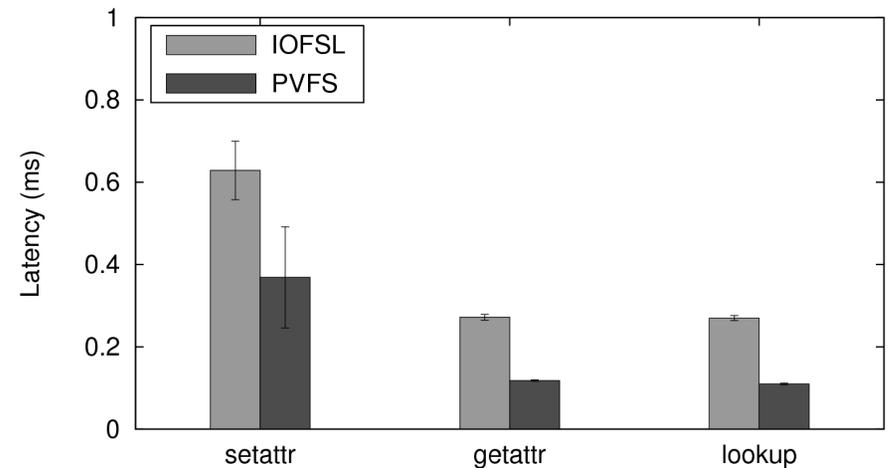
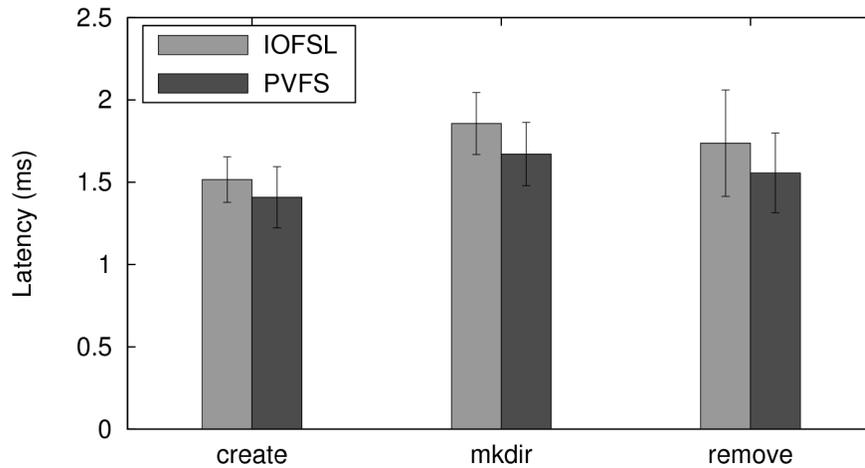
# Prototype test environment



- Experiments performed on a Linux cluster with gigabit Ethernet
- PVFS server with local storage, 16 application processes
- Interpose I/O forwarding in between application and file system
- Study overhead introduced by extra indirection at small scale
  - Additional network link
  - Protocol overhead (encoding/decoding/translation)
- TODO: make this prettier

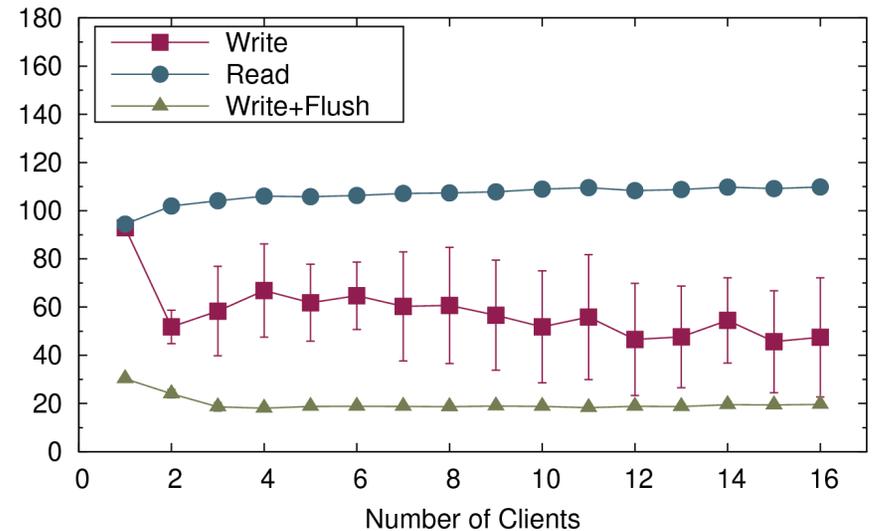
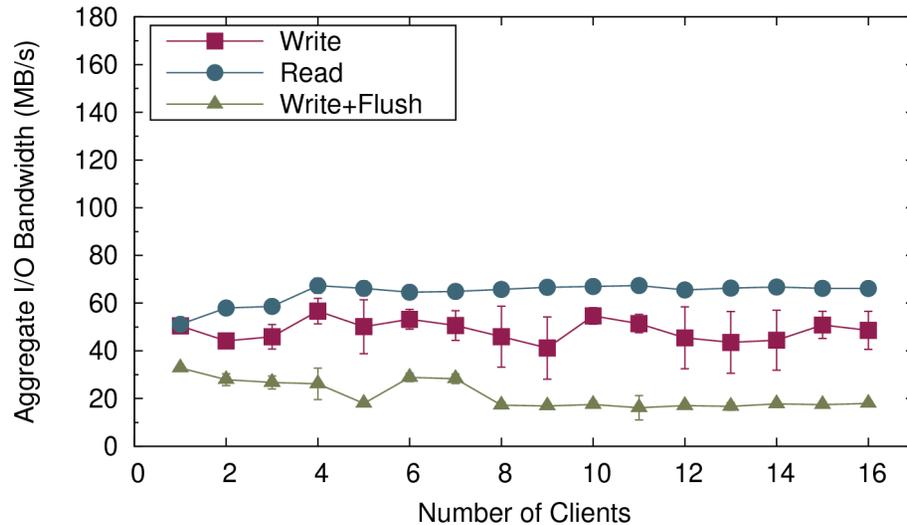


# IOFSL Metadata Latency



- TODO: revise this text
- IOFSL latency is 0.2 ms to 0.3 ms more than that of direct PVFS, fixed cost
- Explain source of overhead
- Higher relative impact on low-latency operations
- Check paper for more stuff
- Different scale in each graph

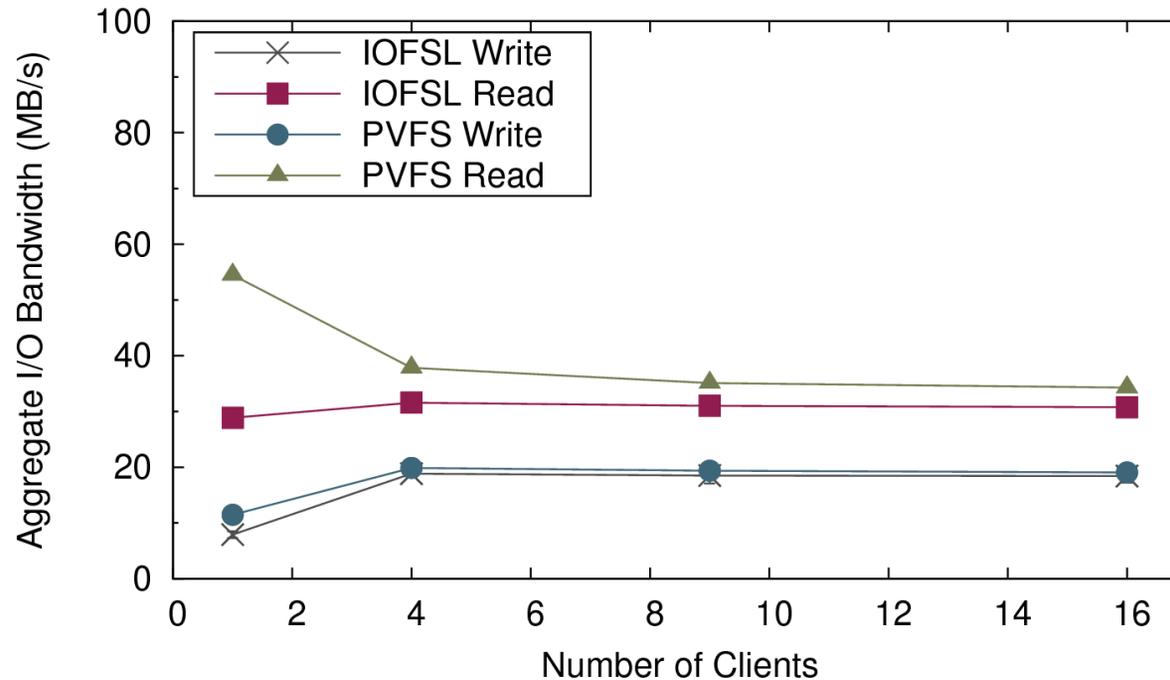
# MPI-IO Performance



- TODO: revise text
- find out if this is using zoid ADIO
- ROMIO perf is an MPI-IO benchmark
- Small number of clients saturate the network
- ROMIO perf data is small enough to be cached in memory (8 MB/client)

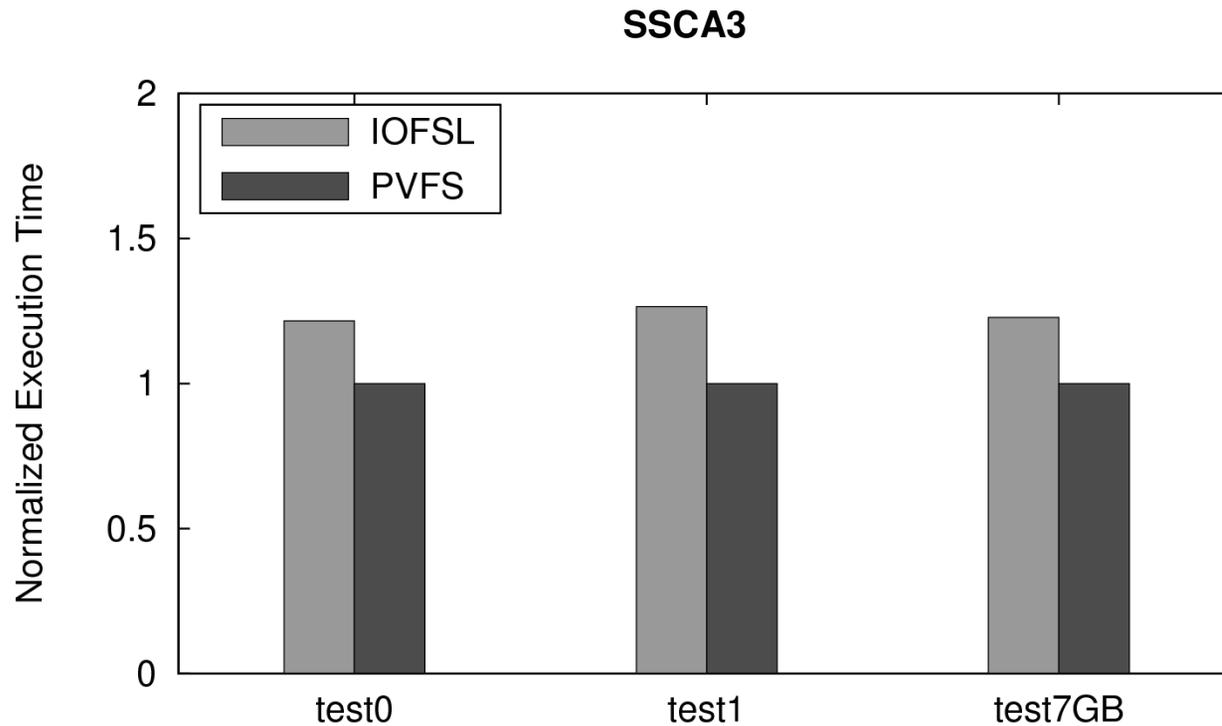
# Application Performance

## BTIO Class C



- TODO: revise text
- BTIO: I/O bandwidth is limited by disk bandwidth

# Application Performance



- TODO: revise text
- SSCA3: IOFSL overhead (21% - 26%)
- Single client
- SSCA modifications

# Conclusions



# Ongoing research

- TODO: what to say here? New prototypes, public release, looking at scalability, aggregation, caching, etc.?



# Questions?

Speaker: Phil Carns ([carns@mcs.anl.gov](mailto:carns@mcs.anl.gov))

Web site:

<http://www.mcs.anl.gov/research/projects/iofsl>

