ILLINOIS INSTITUTE OF TECHNOLOGY

Argonne NATIONAL LABORATORY

Leveraging Burst Buffer Coordination to Prevent I/O Interference

Anthony Kougkas
akougkas@hawk.iit.edu
Matthieu Dorier, Rob Latham, Rob Ross, Xian-He Sun

12th eScience 2016 IEEE INTERNATIONAL CONFERENCE

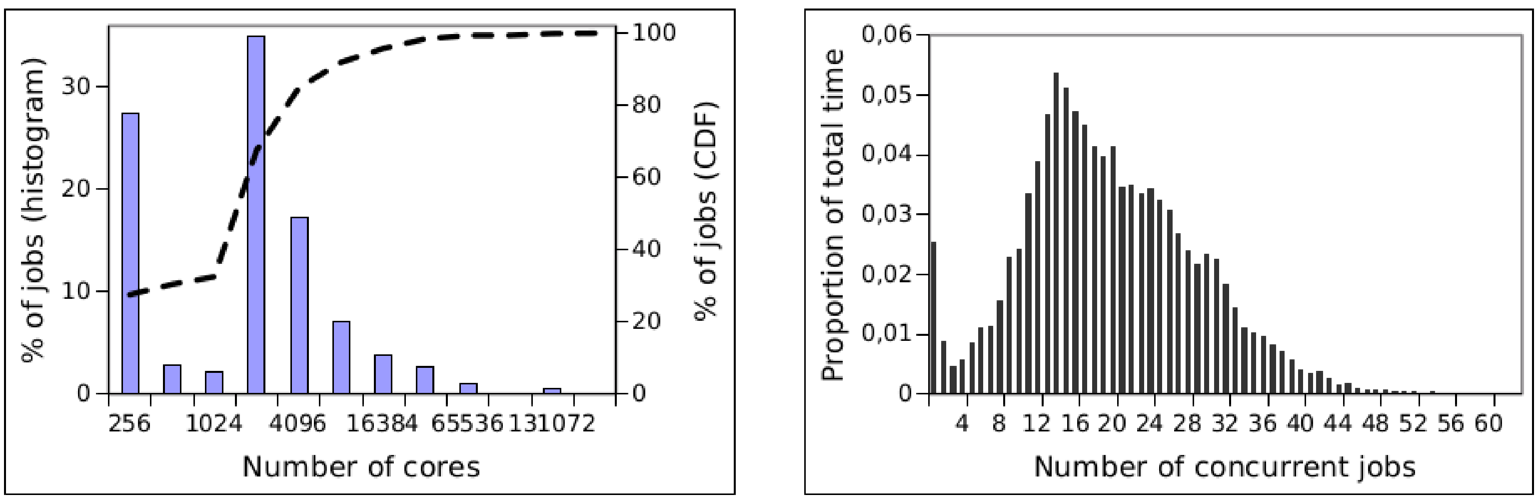
Wednesday, October 26th
Baltimore, USA

Outline

- Introduction and Background
- Methodology
- Design and Implementation
- Experimental Results
- Conclusions

HPC machines run multiple applications concurrently!

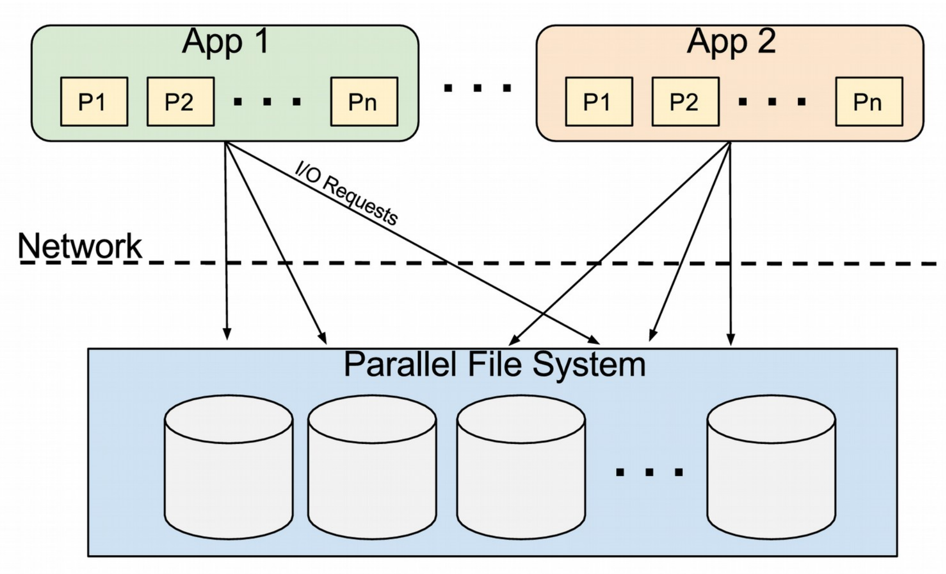
- Capability vs Capacity supercomputers
- Case Study (Argonne Intrepid):
  - Half of the jobs run on less than 2048 cores(1.25% of the full system).
  - Also half the system time was used by jobs smaller than 2048 cores.



(a) Distribution of job sizes (b) Number of concurrent jobs

Figure source: Calciom:Mitigating I/O Interference in HPC Systems through Cross-Application Coordination, M.Dorier et al., IPDPS 14

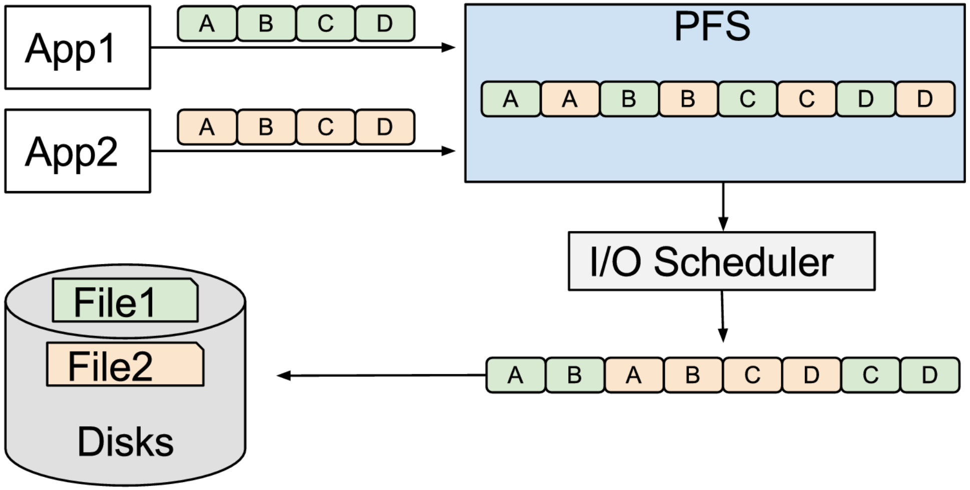
What is I/O interference?



- Network contention at the level of each storage server.
- Poor scheduling decisions within the storage service leading to different servers servicing requests from distinct applications in a different order.
- Additional disk-head movements when interleaved requests reach the same storage device.

Serializing of requests (usually FCFS order)

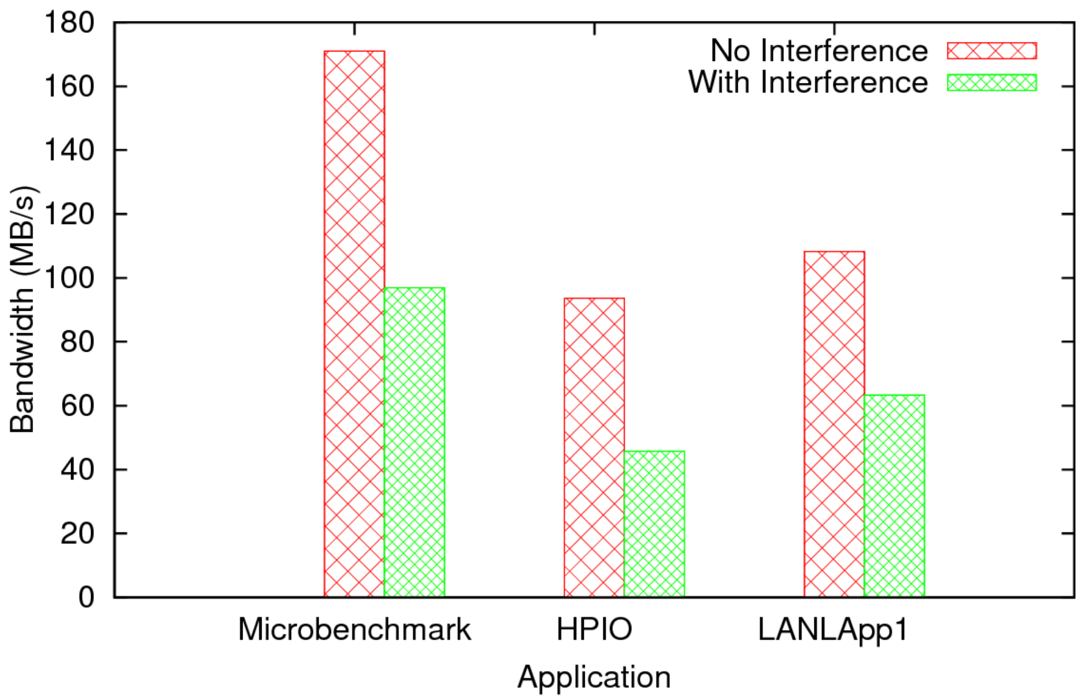
I/O performance degradation due to applications' INTERFERENCE



Cross-application I/O interference effects

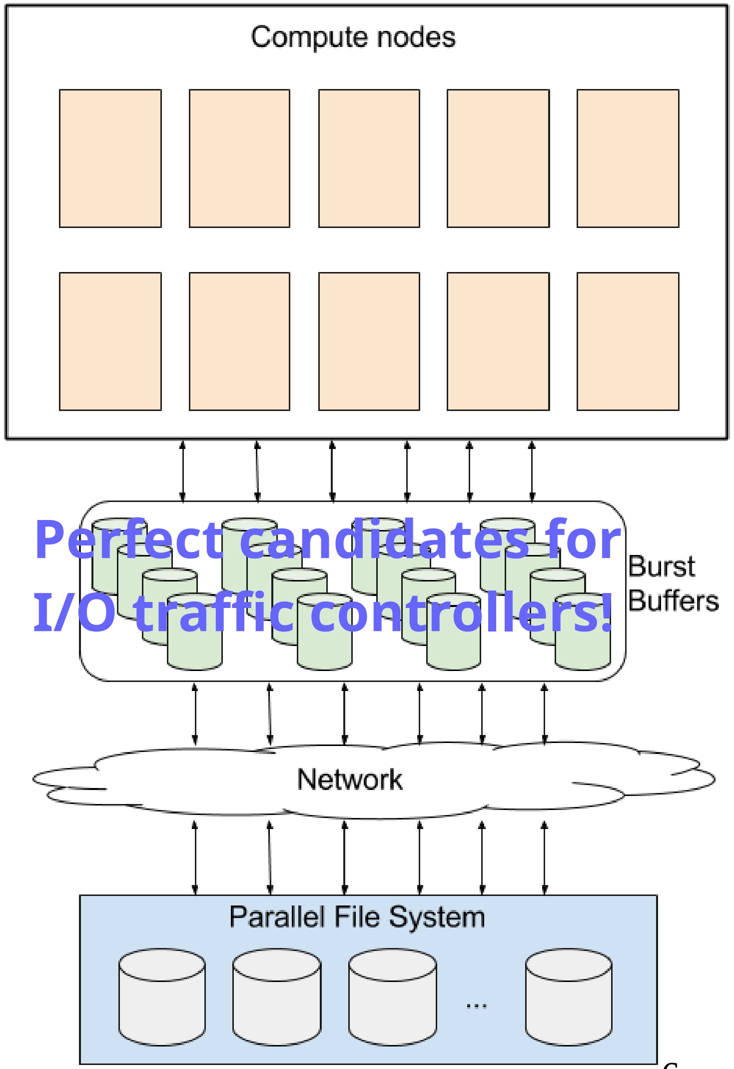
- Significant performance degradation (as low as 50%)
- Lower global I/O efficiency
- Applications experience higher I/O latency

There is a better way!



What is a burst buffer?

- Burst buffers are an intermediate storage tier located between compute nodes and the underlying storage system.
- Main goal: to quickly absorb I/O requests from the computing elements and asynchronously issue them to the PFS, allowing the processing cores to return faster to computation.



Outline

- Introduction and Background
- Methodology
- Design and Implementation
- Experimental Results
- Conclusions

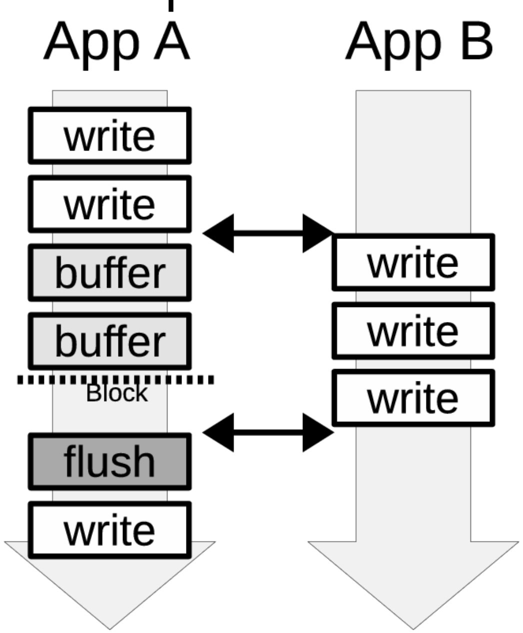
Our approach

- Coordinate data accesses to prevent applications to reach the underlying storage resources at the same time.
- Control I/O accesses by imposing certain I/O policies implemented by the burst buffer layer.
- Solution should be non-invasive to the applications.
- Three new strategies to prevent I/O interference.
- By solving I/O interference, our solution promises:
  - higher global I/O efficiency
  - better performance

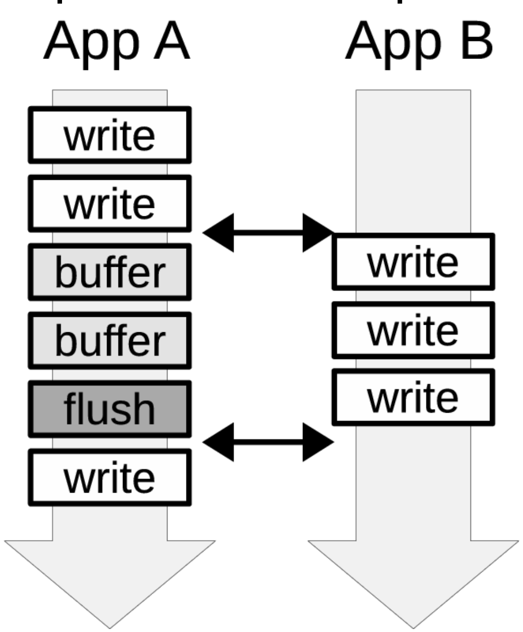
Strategy 1

- App A gets interrupted by another App B and stages the I/O.
- Two variants:

App A blocks if necessary and flushes only when App B has completed its I/O operations.



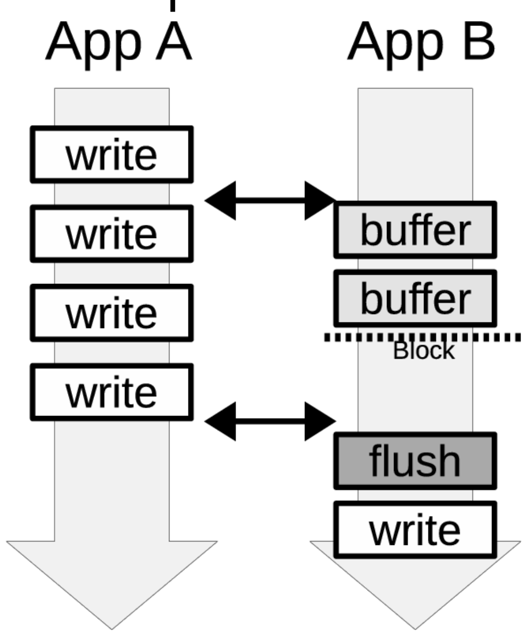
App A flushes its buffer when it has no more available work even if App B has not completed its I/O operations.



Strategy 2

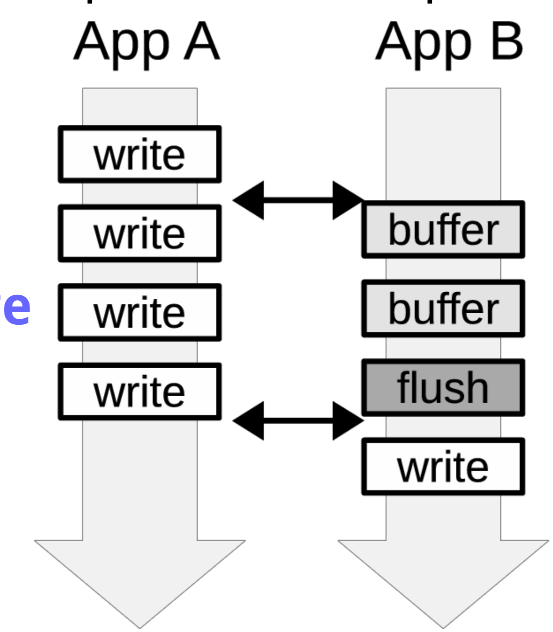
- App A never gets interrupted. App B upon arrival stages its I/O.
- Two variants:

App B blocks if necessary and flushes only when App A has completed its I/O operations.



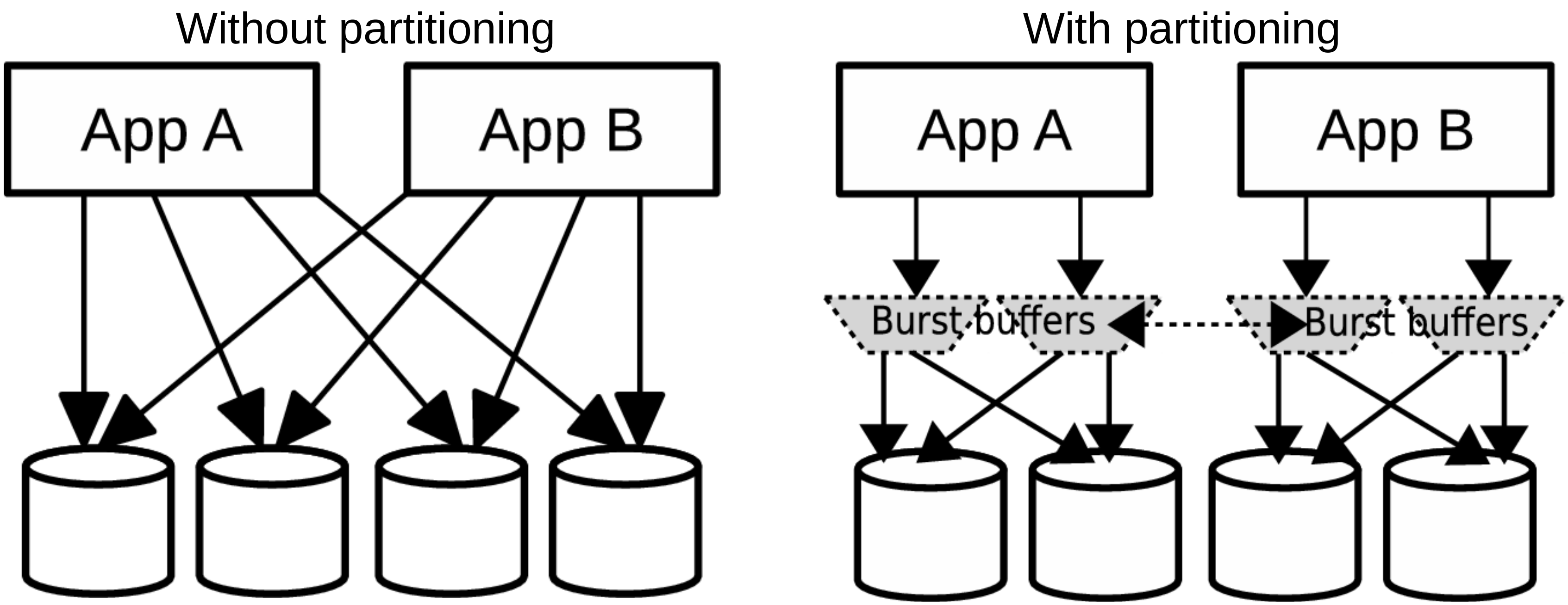
It is a question of which application enjoys exclusive access to the storage resources!

App B flushes its buffer when it has no more available work even if App A has not completed its I/O operations.



Strategy 3

- Partition the parallel file system's servers into distinct subsets.
- Two modes: static and dynamic partitioning.



Outline

- Introduction and Background
- Methodology
- Design and Implementation
- Experimental Results
- Conclusions

Design: High-level architecture

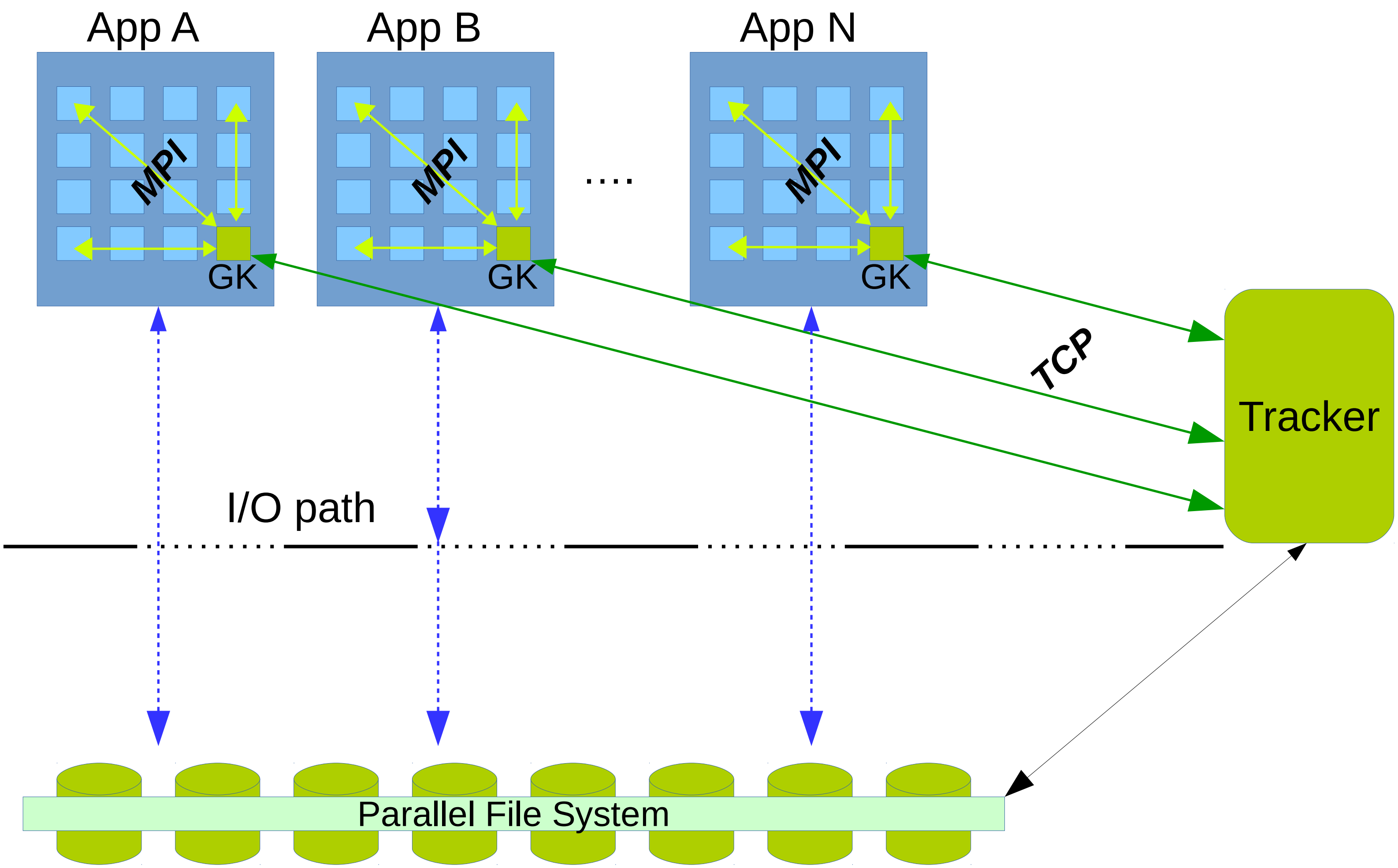
<u>Key components:</u>

Gatekeepers → Collect info about I/O patterns and represent the app

Tracker → Collects info from all GK and imposes the policies

Selective buffering → sets on or off the data staging

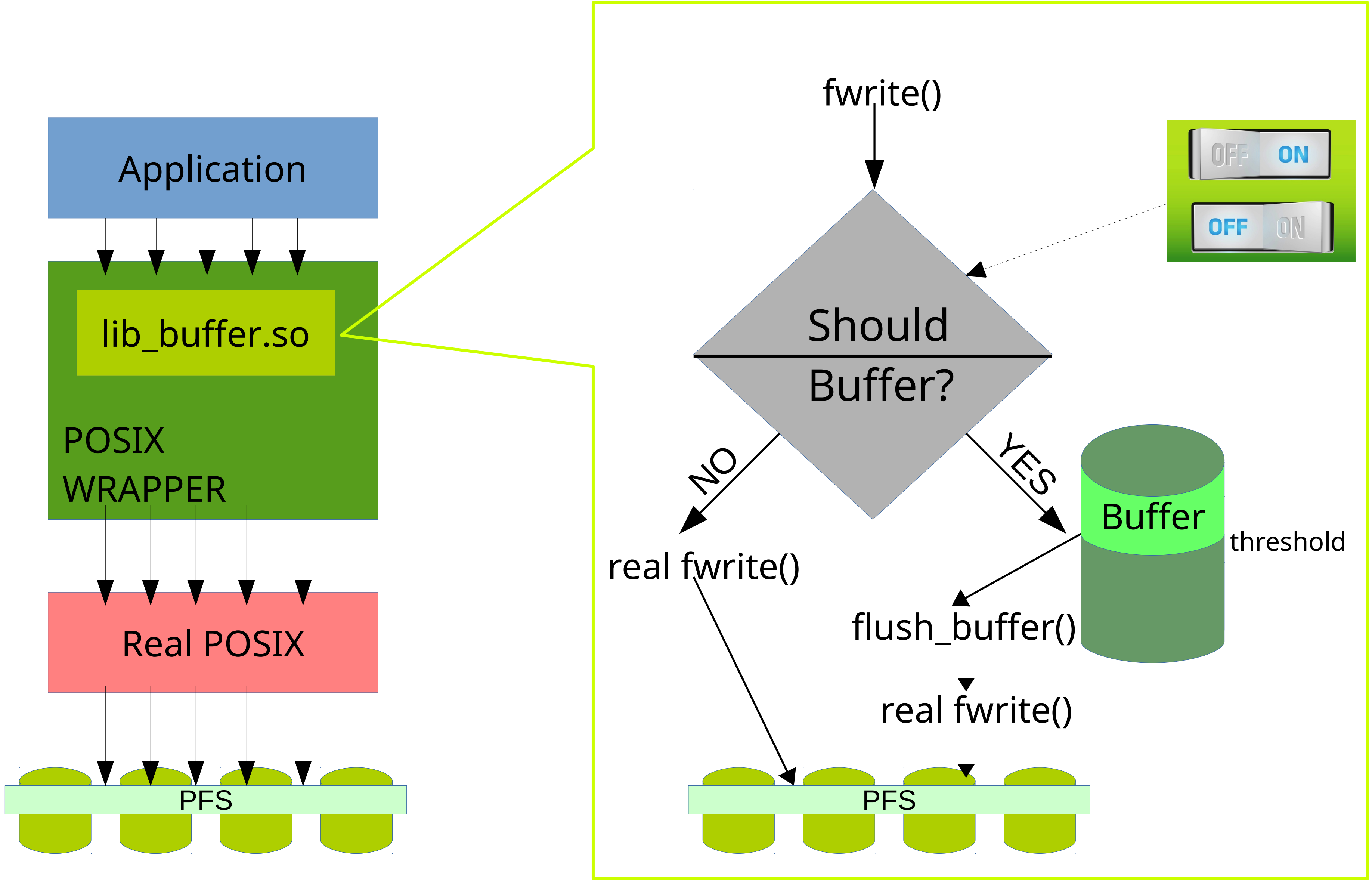
PFS module → dynamically partitions PFS servers



Implementation overview

- Basic Buffered I/O (BBIO): a user-space buffering system under POSIX and LibC interfaces.
- BBIO library can be linked to any code statically or dynamically if preloaded.
- POSIX and MPI function wrappers redirecting I/O traffic.
- Simple API:
  - BBIO_Init(), BBIO_Finalize()
  - BBIO_Enable(), BBIO_Disable()
  - BBIO_Flush(), BBIO_On_flush()
- Find it in https://bitbucket.org/mdorier/bbio

Implementation overview



Outline

- Introduction and Background
- Methodology
- Design and Implementation
- Experimental Results
- Conclusions

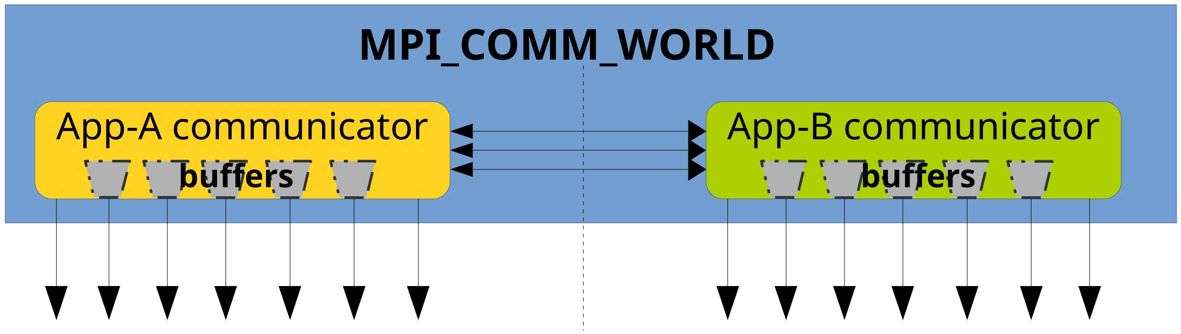
Experimental setup

Hardware:

- 65-node Linux cluster
- 8-cores and 8GB of RAM
- HDD + SSD on PCIe
- InfiniBand + 1Gb Ethernet
- Exclusive access (1 user)

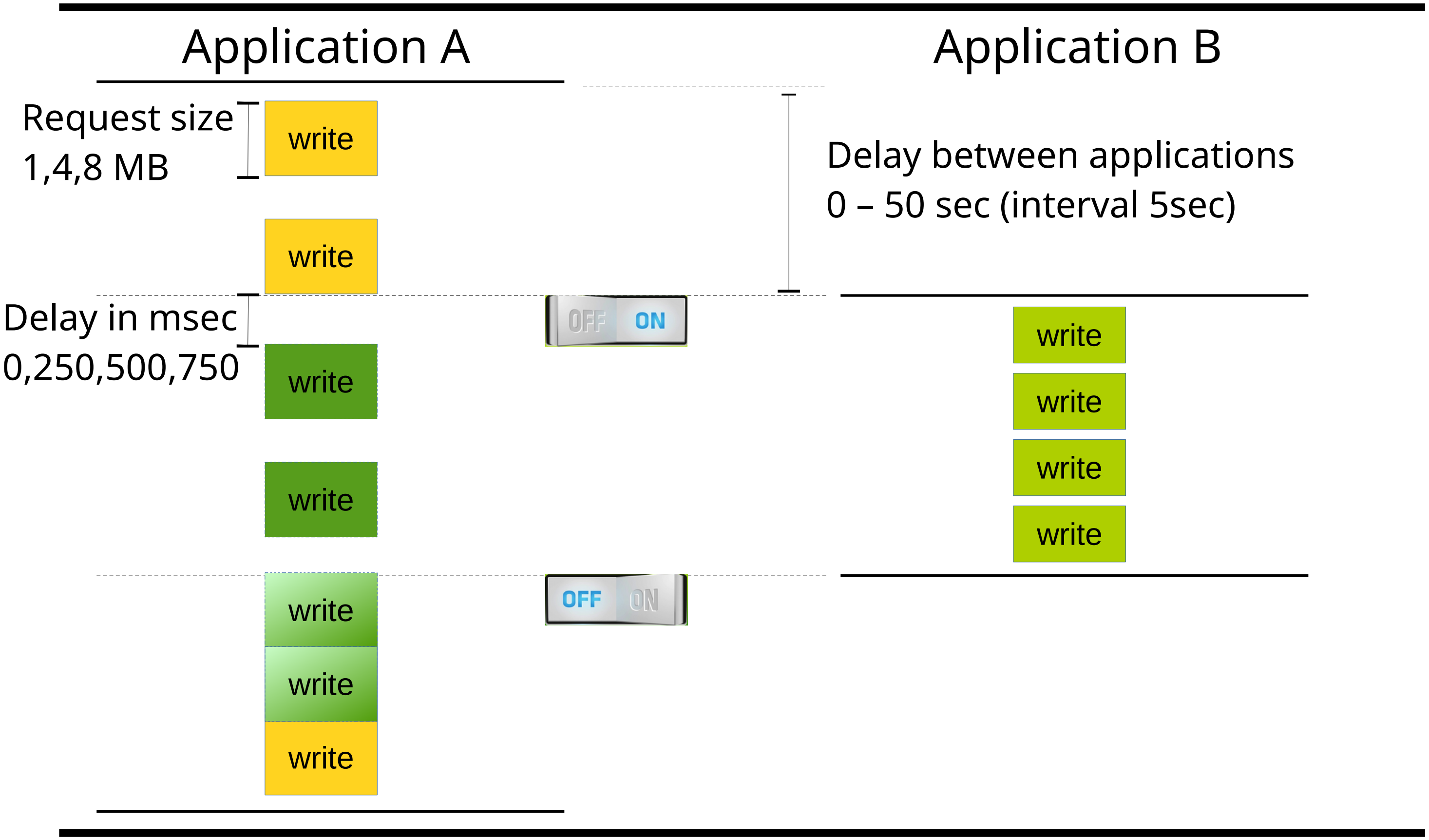
Software:

- Ubuntu Server 12.04
- OrangeFS 2.9.2
- Gcc 4.8 and MPICH 3.1.4
- Our own micro-benchmark
- CM1 and LANL_App1&2



Design of our micro-benchmark

- Example of the benchmark for Policy 1:



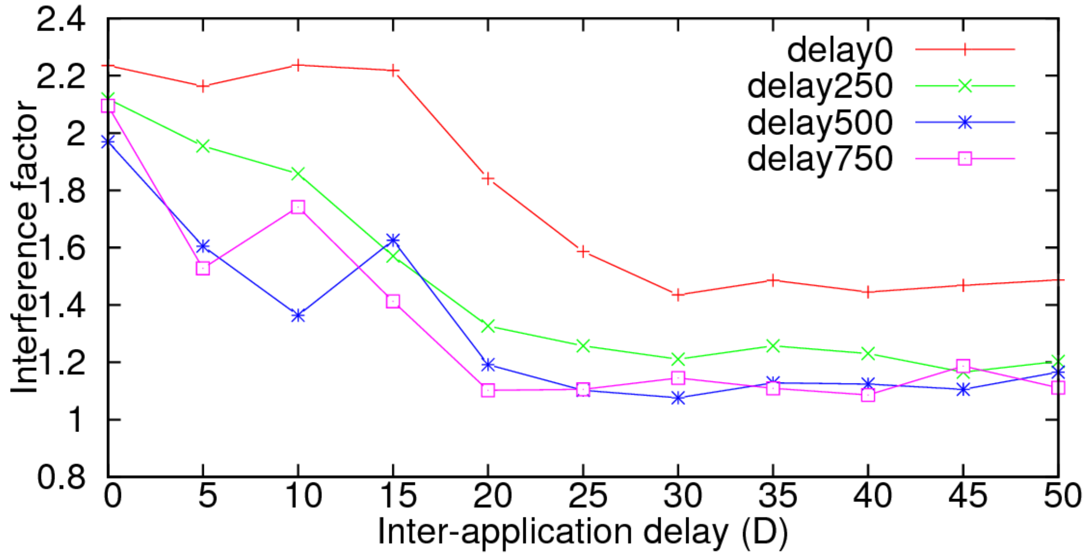
- 256 MPI ranks
- 32 MB per rank
- 8 pvfs2 servers
- 128MB buf/node
- 10x repetitions

Interference definition:

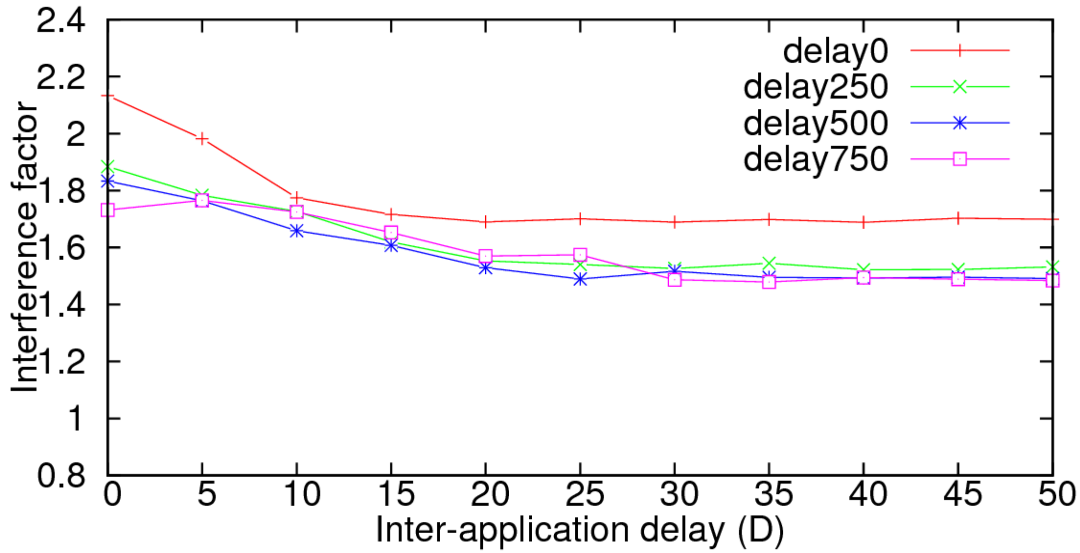
$$I = \frac{T}{T_{alone}} > 1$$

Δ-graphs: App A&B with no policy (default)

Application A

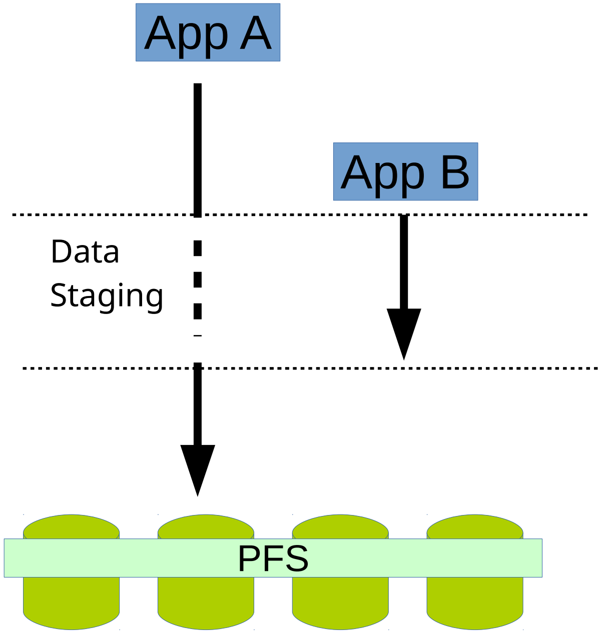


Application B

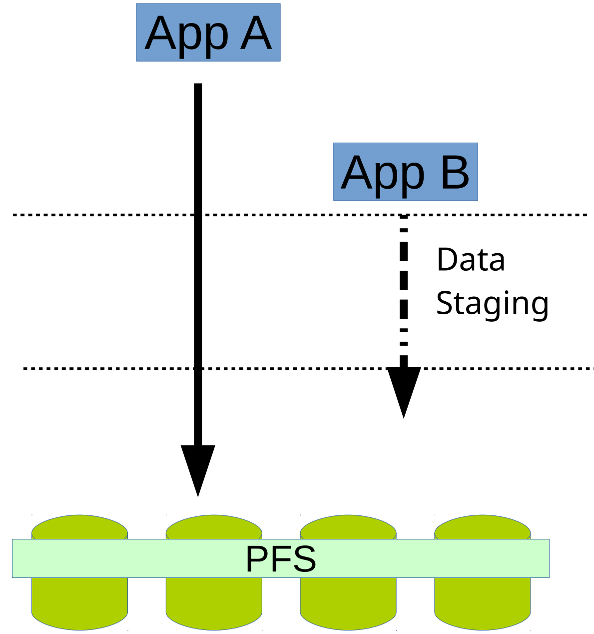


Policies (quick recap)

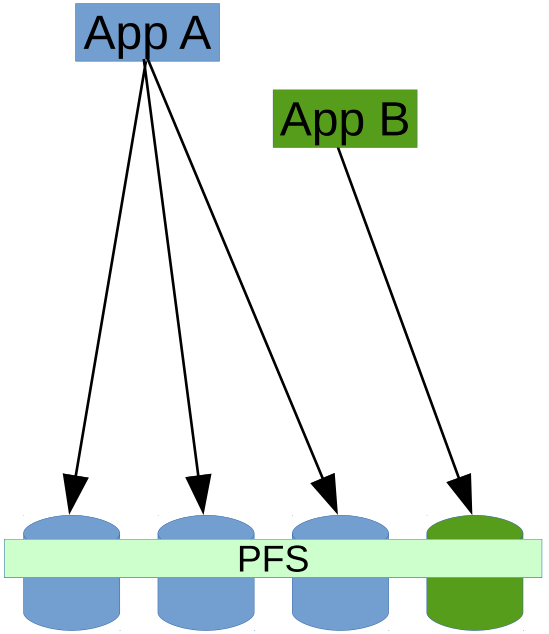
- Policy 1: Application A is running and gets interrupted by Application B



- Policy 2: Application A is running and never gets interrupted



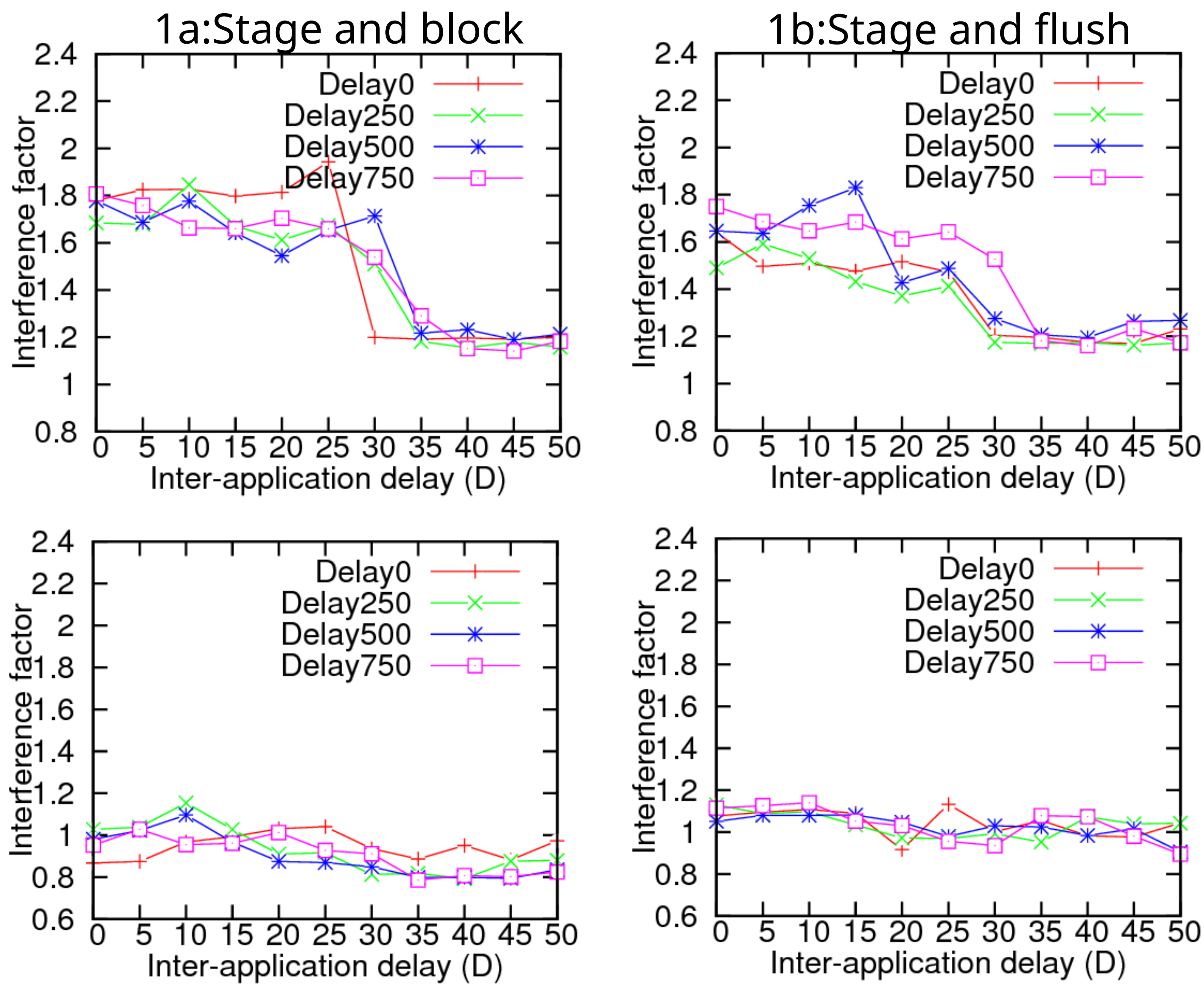
- Policy 3: Application A is running but allowed to access only specific parts of the Parallel File System



Δ-graphs: App A&B with policy 1

Application A

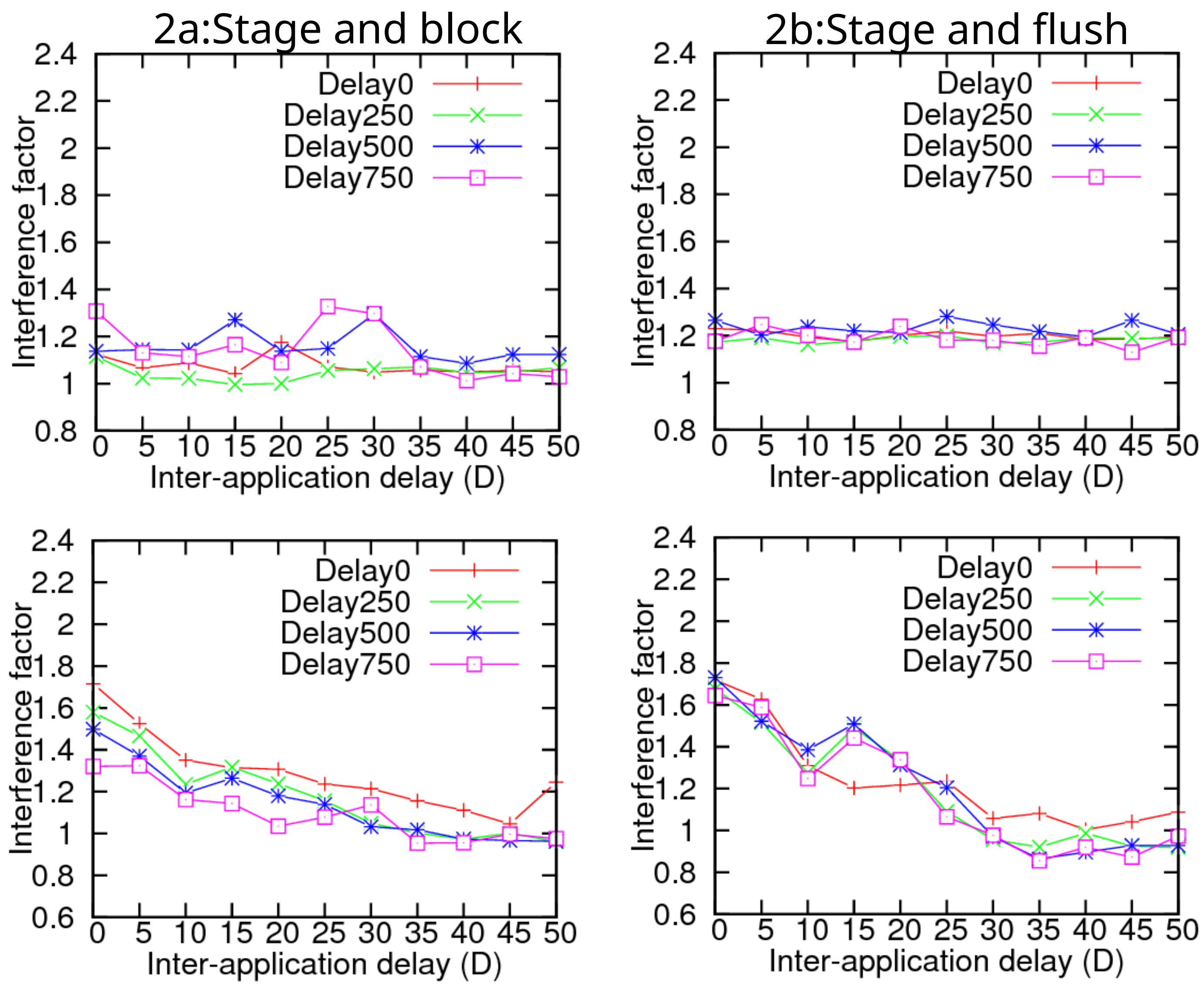
Application B



Δ-graphs: App A&B with policy 2

Application A

Application B

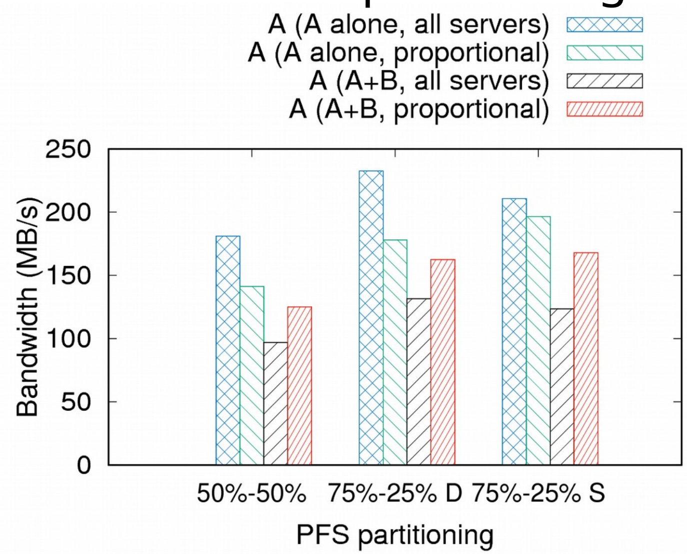


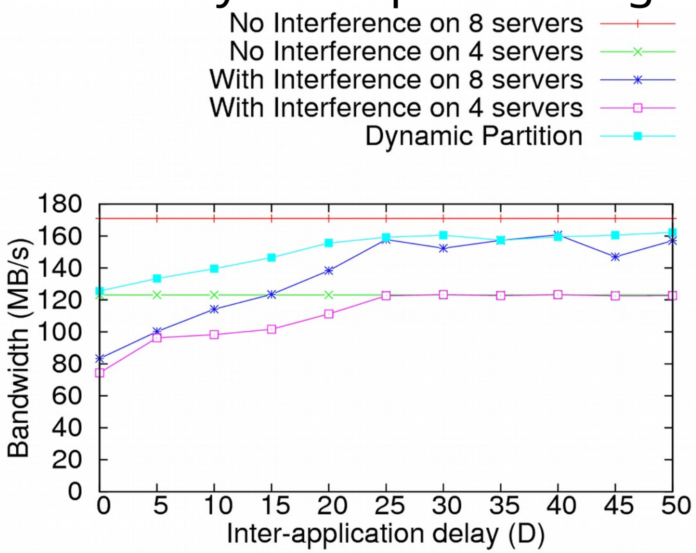
App A&B with policy 3

3a:Static partitioning

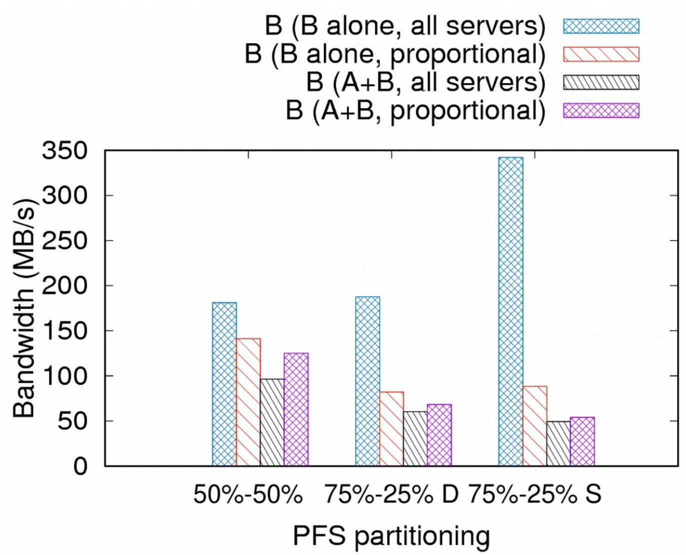
3b:Dynamic partitioning

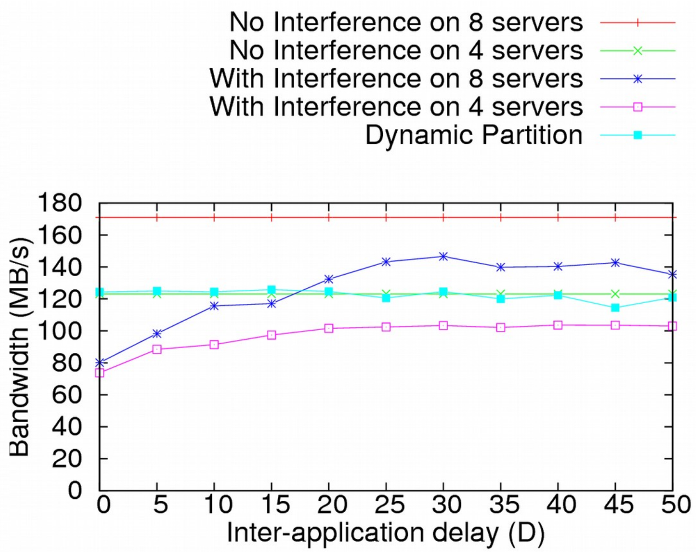
Application A





Application B

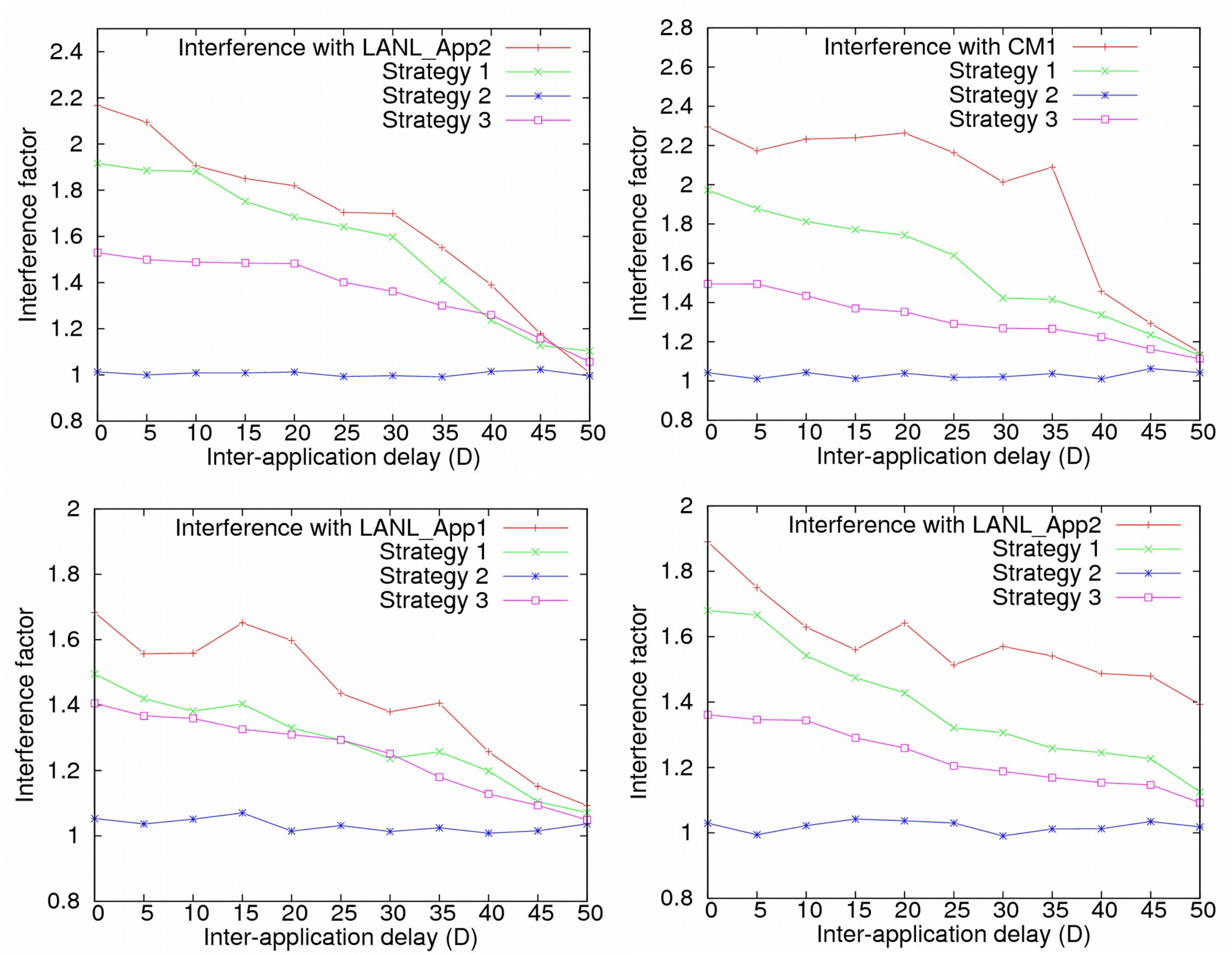




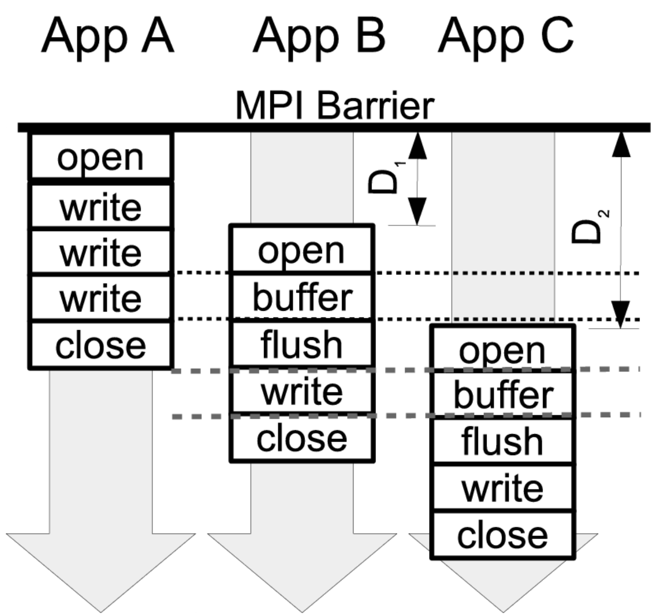
Δ-graphs: Real applications

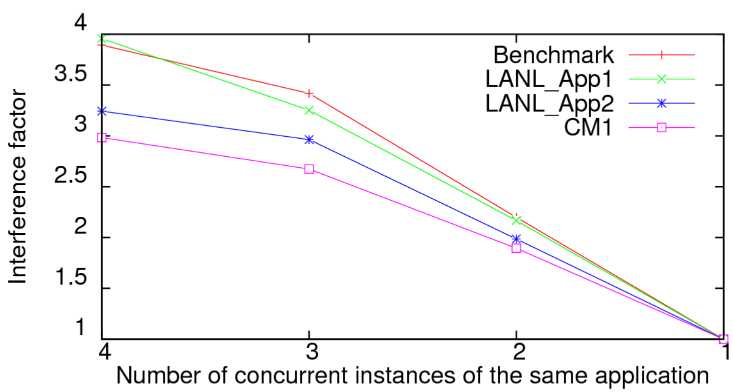
LANL_App1 interfering with LANL_App2 and CM1

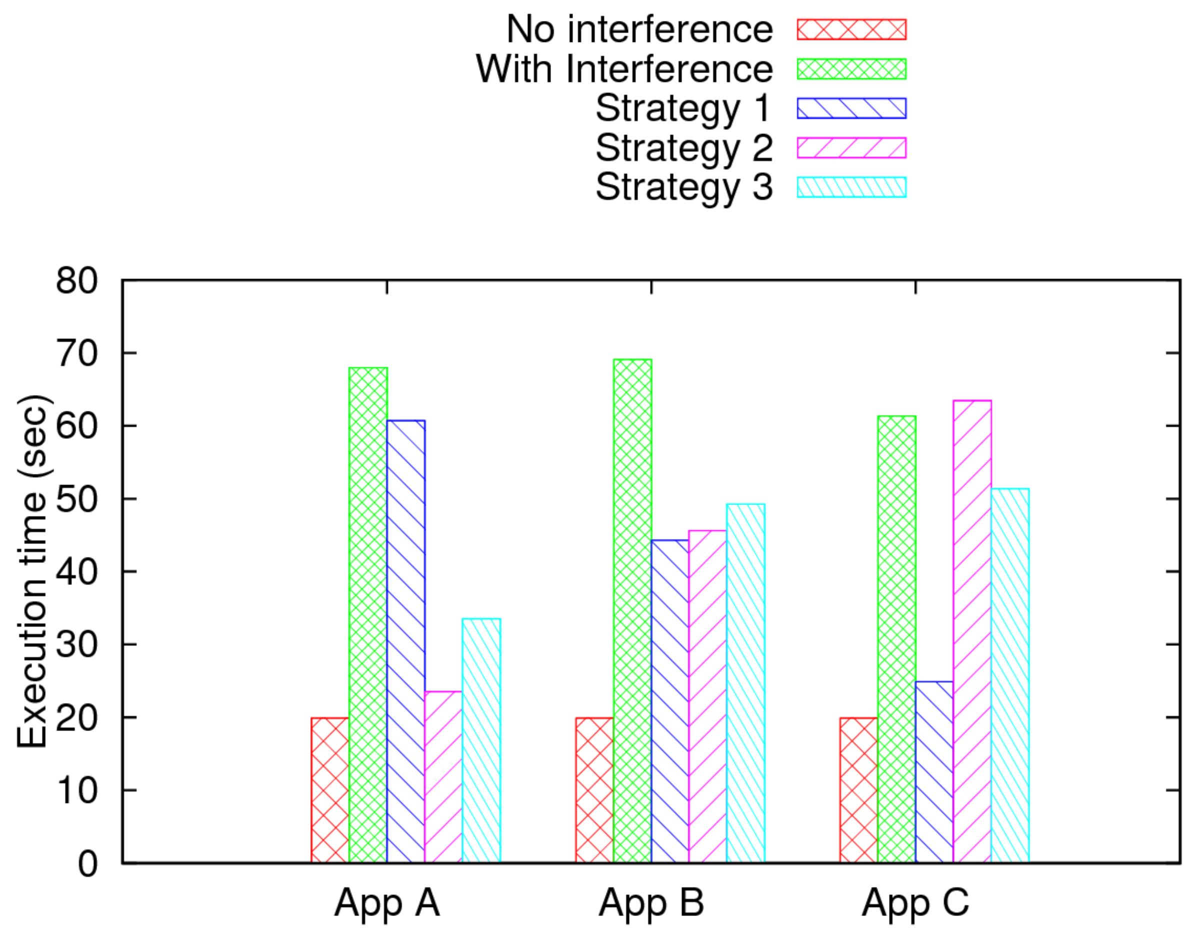
CM1 interfering with LANL_App1 and LANL_App2



Scaling study (more than 2 apps)







Outline

- Introduction and Background
- Methodology
- Design and Implementation
- Experimental Results
- Conclusions

Conclusions

- We demonstrated the negative effects of I/O interference when multiple applications are concurrently executing in an HPC environment.
- We proposed three I/O Policies to mitigate the performance degradation.
- We developed BBIO library which helps impose the proposed I/O policies.
- Experimental results showed that we can achieve higher performance **up to 2x** depending on the selected policy.

Q & A

Leveraging Burst Buffer Coordination to Prevent I/O Interference

Anthony Kougkas

akougkas@hawk.iit.edu

12th eScience 2016 IEEE INTERNATIONAL CONFERENCE