vPFS: Bandwidth Virtualization of Parallel Storage Systems

Yiqi Xu, Dulcardo Arteaga, Ming Zhao Florida International University
Yonggang Liu, Renato Figueiredo University of Florida
Seetharami Seelam IBM T.J. Watson Research Center

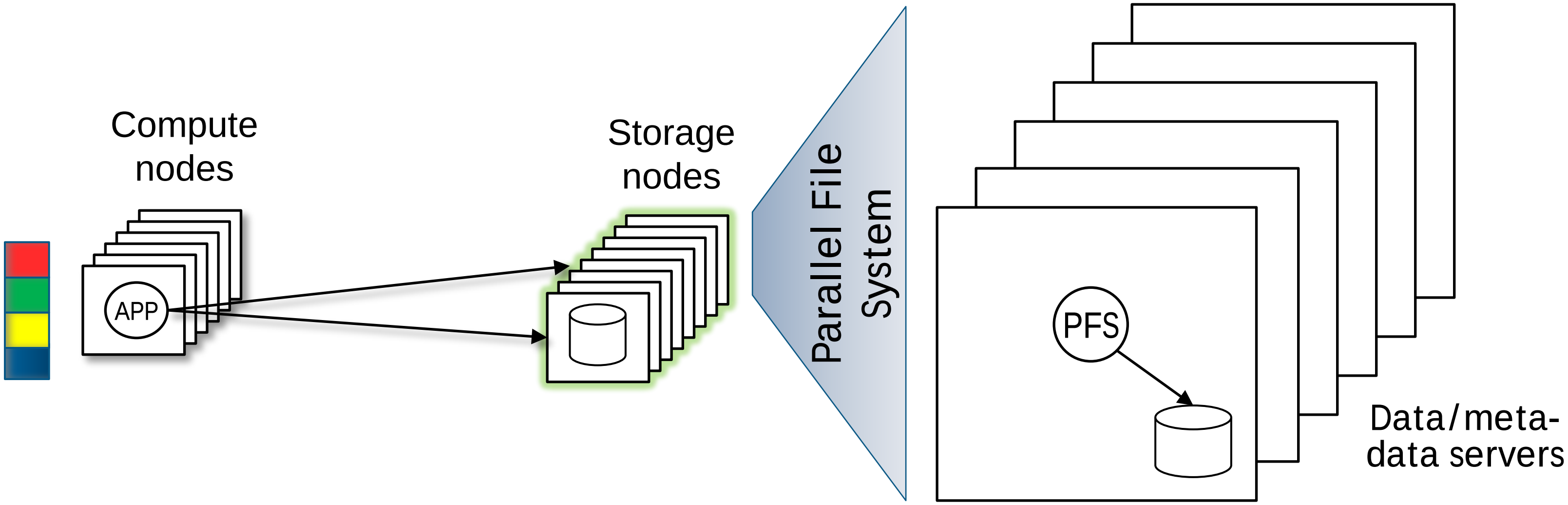
FLORIDA INTERNATIONAL UNIVERSITY

VISA

Background

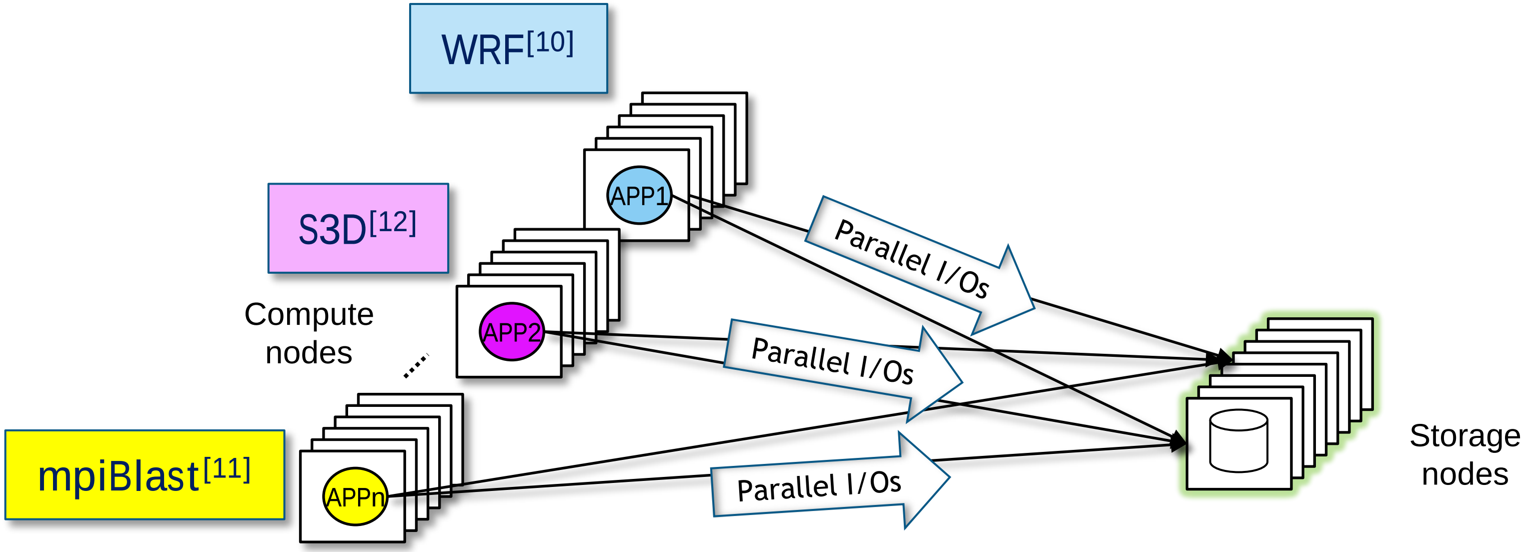
- High Performance I/O supports High Performance Computing (HPC) systems
  - HPC applications become increasingly data intensive
  - Important to match the parallelism of HPC compute nodes
- Parallel File Systems
  - Widely used in HPC systems
    - PVFS2[1], PanFS[2], GPFS[3], Lustre[4], etc.
  - Use parallel I/Os to achieve high throughput

Background



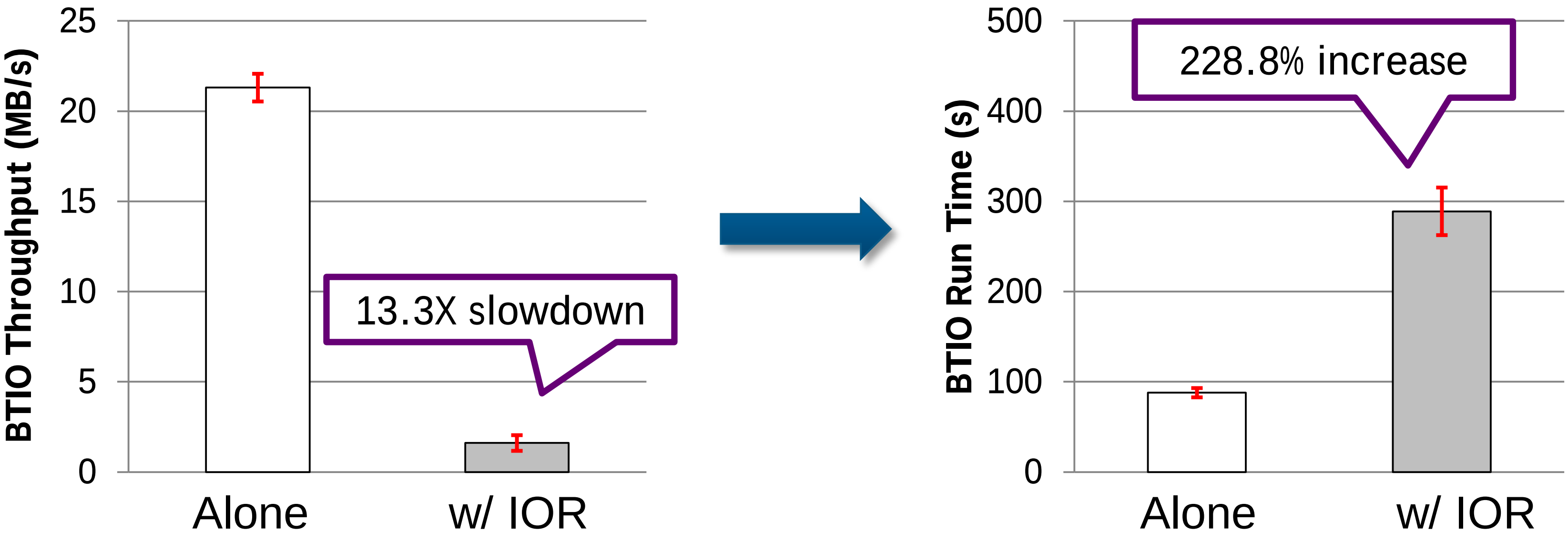
- Parallel File System
  - o Striped I/Os across multiple storage nodes
  - o Aggregated throughput for high-performance I/Os
- Components
  - o Server side: data server daemon, meta-data server daemon
  - o Client side: MPI-IO[14] library, client daemon

Motivation



- Parallel storage is commonly shared
    - Applications have different I/O demands — storage nodes cannot recognize them
    - Their I/Os interfere with each other — storage nodes cannot isolate them

Motivation — BTIO[9] vs. IOR[8]



- BTIO performance severely impacted by IOR
  - I/O time increases > 10x; Total runtime increases > 200%
- Resulted by lack of QoS on the parallel storage

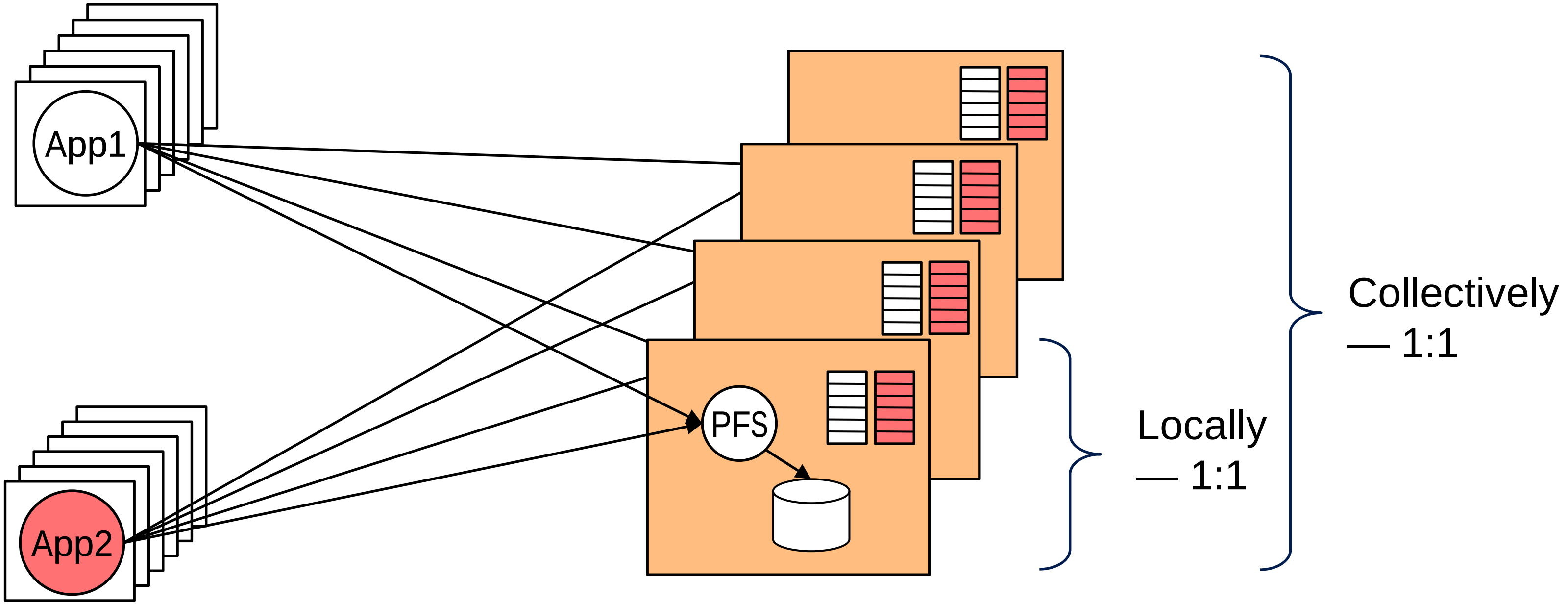
Overview

- Goal
  - Achieve proportional sharing of parallel file system storage
- Challenges
  - Transparent support for existing HPC systems
    - Virtualized PFSes
  - Per-application parallel I/O scheduling
    - Distributed scheduling
  - Scalable implementation of proportional sharing
    - Low-cost synchronization

Outline

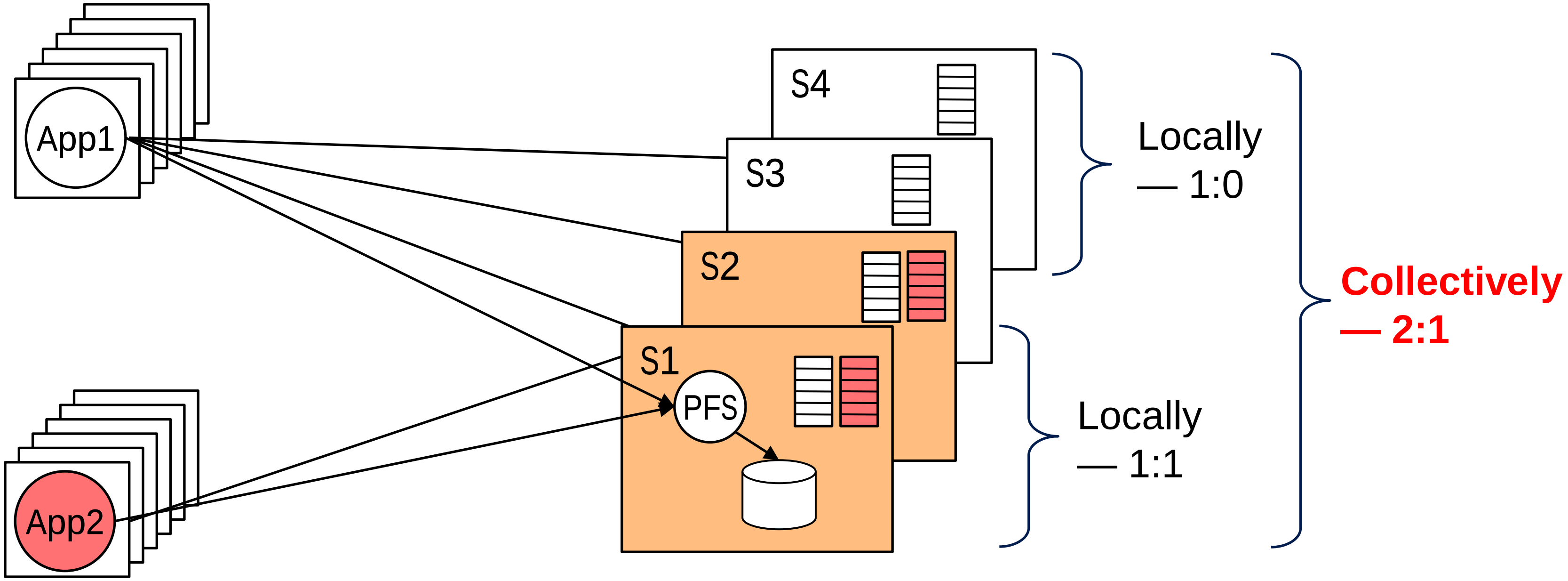
- Background, Motivation, Overview
- Challenges for Total-Service Proportional Sharing
- Solution — vPFS Virtualization and Scheduling
- Experimental Evaluation
- Conclusions

Proportional Sharing on Storage



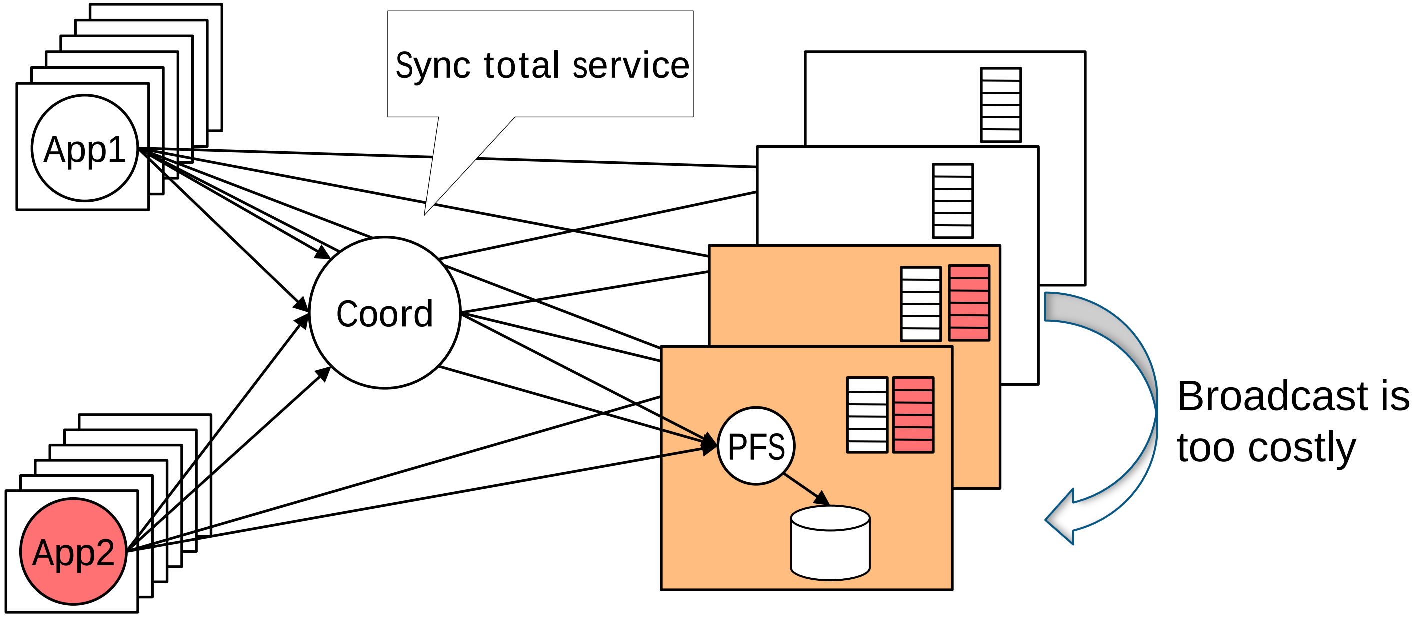
- Local scheduling according to global sharing ratio
- Multi-node aggregated throughput also conforms to global share ratio
  - Assumption: application file layouts are the same

Total-Service Proportional Sharing



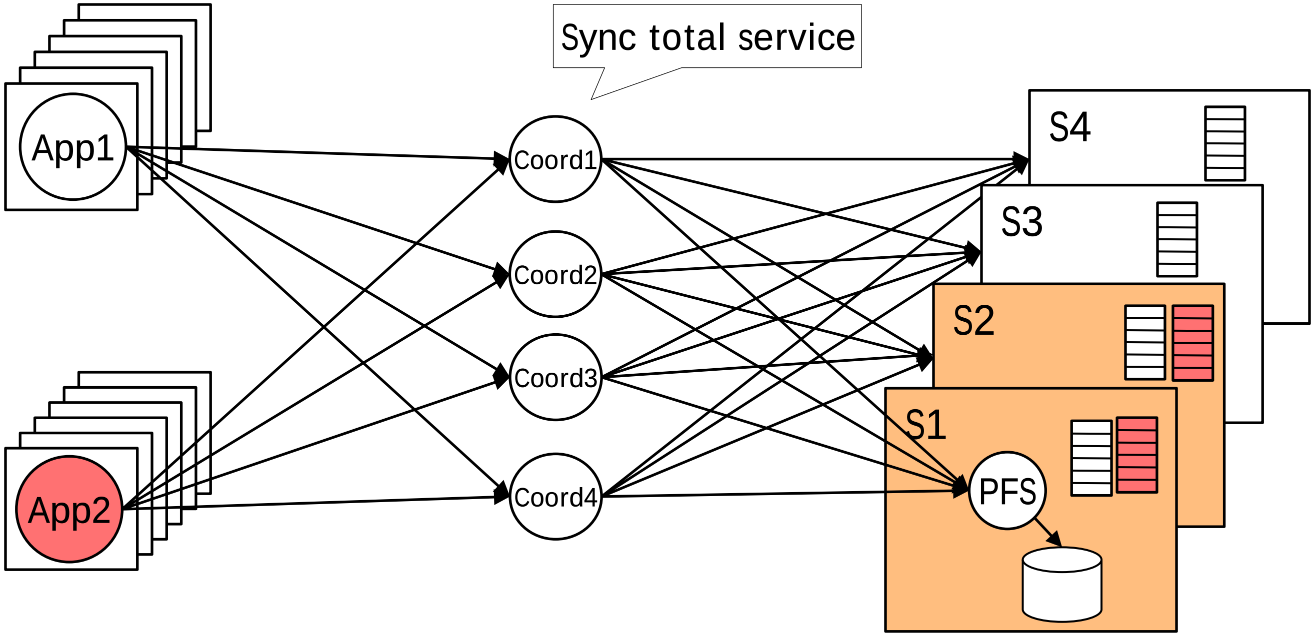
- Local proportional sharing algorithms (SFQ(D)[6]) are not enough for total service fairness
- Global synchronization is necessary among local schedulers — distributed SFQ (DSFQ[7])

Limitations of DSFQ on Parallel Storage



- Broadcast-based synchronization is expensive
- A centralized coordinator is not scalable

Limitations of DSFQ on Parallel Storage



- Broadcast-based synchronization is expensive
- A centralized coordinator is not scalable
- Distributed coordinators do not fit HPC architecture
  - HPC apps access data using predetermined layout

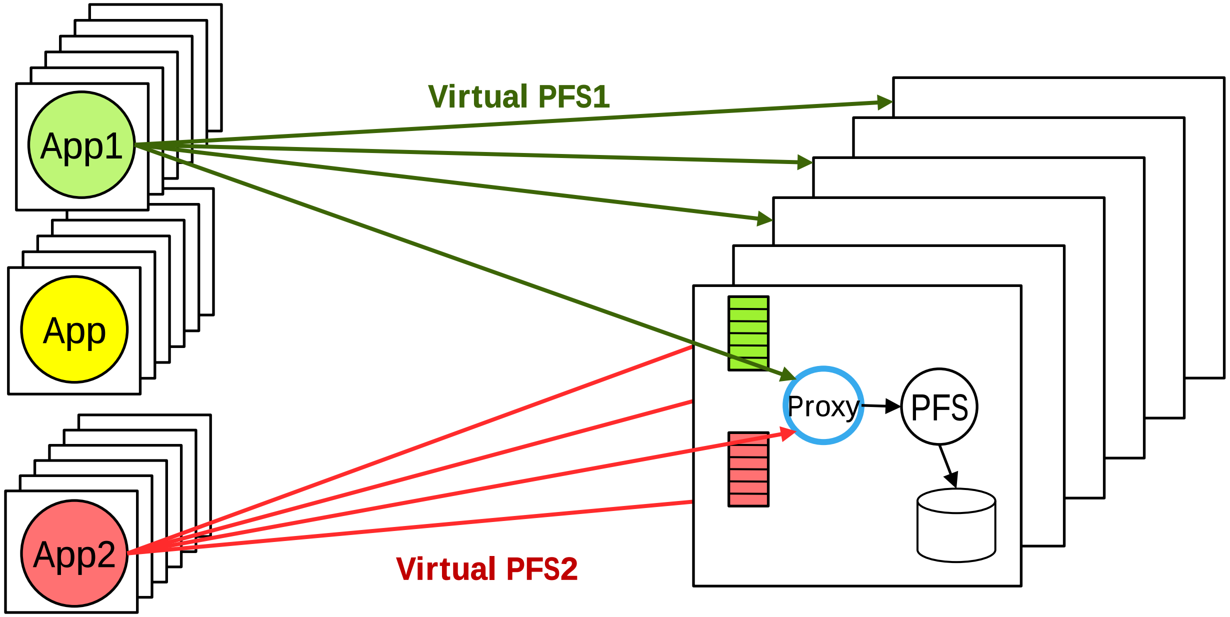
Outline

- Background, Motivation, Overview
- Challenges for Total-Service Proportional Sharing
- Solution — vPFS Virtualization and Scheduling
- Experimental Evaluation
- Conclusions

Solution – vPFS

- Enable per-application virtual PFSes
- Enable distributed scheduling upon the vPFS framework with low-cost synchronization
- Achieve total-service proportional sharing across parallel storage servers
- Support flexible study of different schedulers on parallel file system storage

vPFS — Virtualization Layer



- Create virtual PFSes by proxy-based interposition
- Capture and differentiate application I/Os
- Re-order and dispatch according to QoS requirements

vPFS — Scheduling

- Implemented Schedulers
  - SFQ(D)[6] local proportional sharing
  - Threshold-driven distributed proportional sharing
  - Layout-driven distributed proportional sharing
- Generic interfaces
  - Flexible to support multiple schedulers of different natures

Naive Synchronization

- Synchronization in parallel scheduling remains unsolved
- Simple broadcast-based synchronization cost:
  - $O(M \bullet A \bullet N^2 \bullet W)$
    - M = sync message size per application
    - A = number of applications
    - W = total bytes serviced
    - N = number of servers
  - Scales with number of servers (*N*)
  - Scales with number of bytes serviced (*W*)

Threshold-driven Synchronization

- Threshold-driven synchronization reduces cost
  - Limits broadcast frequency
    - T = threshold with regard to W
    - Synchronizes only when *W* exceeds *T*
  - Synchronization cost is $O(M \cdot A \cdot N^2 \cdot W/T)$
    - Cost greatly reduced by *T*
    - E.g., 10MB threshold reduces 95% synchronization with 512KB request size
  - With bounded worst-case unfairness
    - Controlled by *T*

Threshold-driven Synchronization

- Unfairness between *f* and *g* bounded[15]:

Share

$$\left|\frac{w_f(t_1,t_2)}{\phi_f}-\frac{w_g(t_1,t_2)}{\phi_g}\right|$$

Depth of scheduler/disk

Number of servers

Max Cost

$$\le \left((D_s+D_d)N_f+1\right)\frac{cost_{f,A}^{\max}}{\phi_f}+\left((D_s+D_d)N_g+1\right)\frac{cost_{g,A}^{\max}}{\phi_g}$$

$$+(D_s+1)\left(\frac{batchcost_{f,A}^{\max}}{\phi_f}+\frac{batchcost_{g,A}^{\max}}{\phi_g}\right)$$

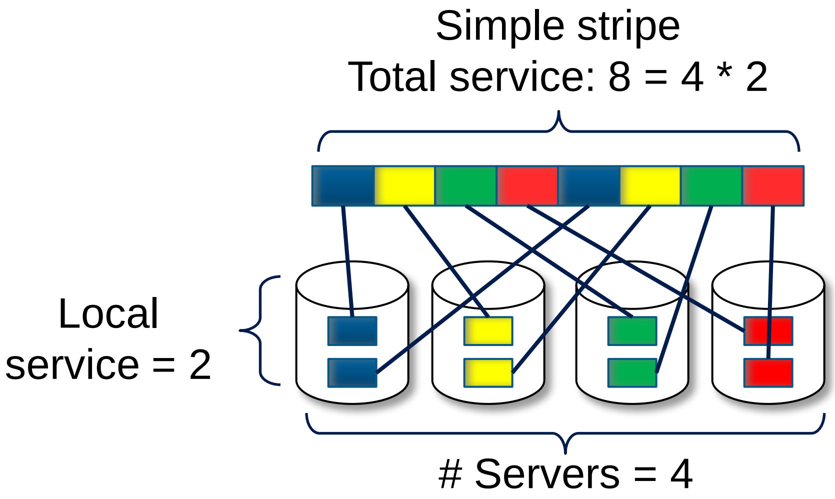
Max total cost between two requests

$$+2(N_f-1)\frac{C_f^{prop}}{\phi_f}+2(N_g-1)\frac{C_g^{prop}}{\phi_g}$$

Synchronization ***T***hreshold

Layout-driven Synchronization

- Threshold-driven synchronization cost still scales quadratically with *N* — $O(M \cdot A \cdot N^2 \cdot W/T)$
- Layout-driven synchronization is proposed
  - Utilizes file layout of each application
  - Transforms global communication into local computation
- Approximate total-service
  - Using local service I/Os
  - Needs file layout information
    - Stripe method
    - Stripe parameters



Layout-driven Synchronization

- Availability of Layout
  - PFS protocol
    - E.g., PVFS2 I/O request header has stripe information
  - Meta-data server
    - Meta-data is generally available
  - Arrival and departure of applications
    - Servers notifies others when it sees the first I/O of an app
- Limitation of Layout
  - Small I/Os that are not evenly distributed on all servers
  - Threshold-driven synchronization works better

Layout-driven Synchronization

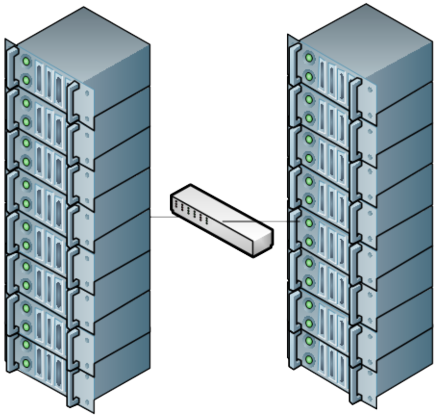
- Synchronization cost further reduced to O(M•A•N)
  - Cost is much lower than threshold-driven scheme
    - Scales only linearly with number of servers (*N*)
    - Independent of total bytes serviced (*W*)
    - Incurs less interference between application I/Os (*W*) and synchronization I/Os (*M•A*)
      - Synchronizes only when application arrives/departs
      - So that layout is available

Outline

- Background, Motivation, Overview
- Challenges for Total-Service Proportional Sharing
- Solution — vPFS Virtualization and Scheduling
- Experimental Evaluation
- Conclusions

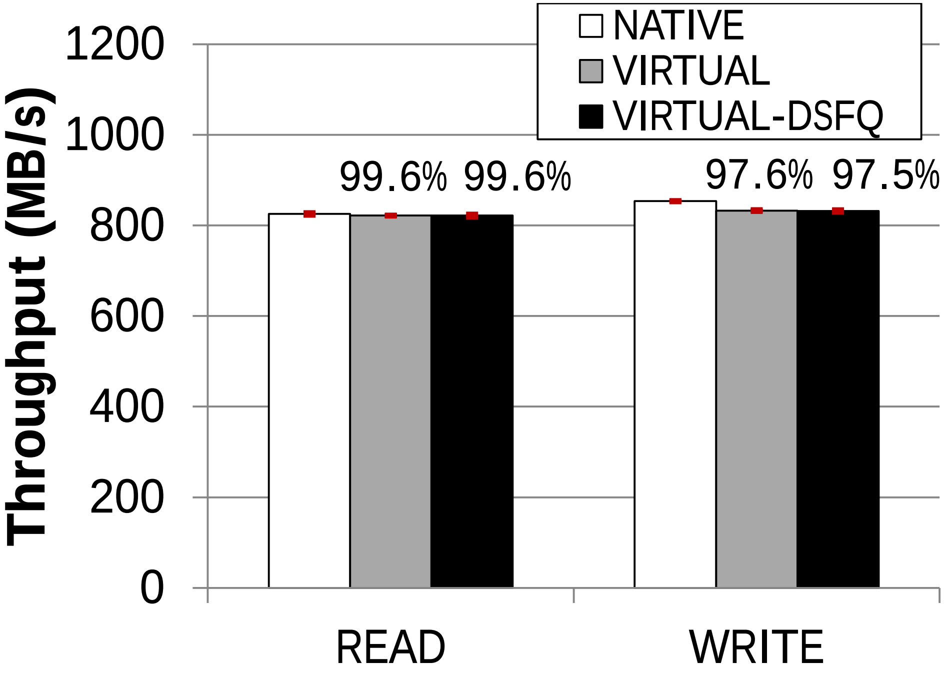
Evaluation

- Hardware
  - 8 Clients & 8 Servers, 1 gigabit switch
- Software
  - PVFS 2.8.2 — up to 96 daemons
  - IOR 2.10.3 — up to 256 processes
  - BTIO 3.3.1-MPI — up to 64 processes



- Experiments
  - Overhead of proxy-based virtualization
  - Effectiveness of total-service proportional sharing
  - Comparison of different synchronization schemes

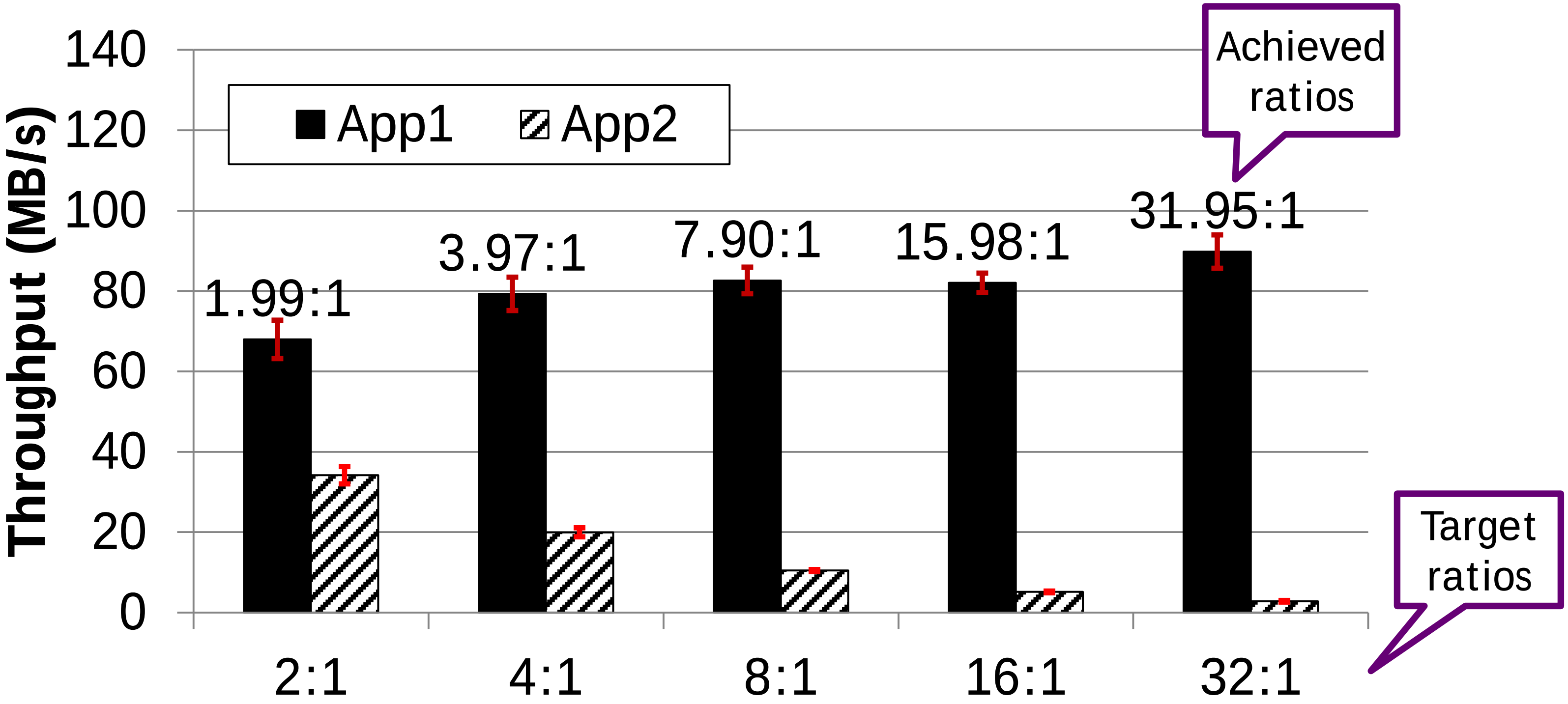
vPFS Overhead



- Comparing 3 cases
  - Native: PVFS only
  - Virtual: PVFS + vPFS
  - Virtual+DSFQ: PVFS + vPFS + DSFQ
- Worst case scenario overhead

- Throughput overhead is below 3%
- CPU and memory overhead is below 1%

2 IORs — Write vs. Write

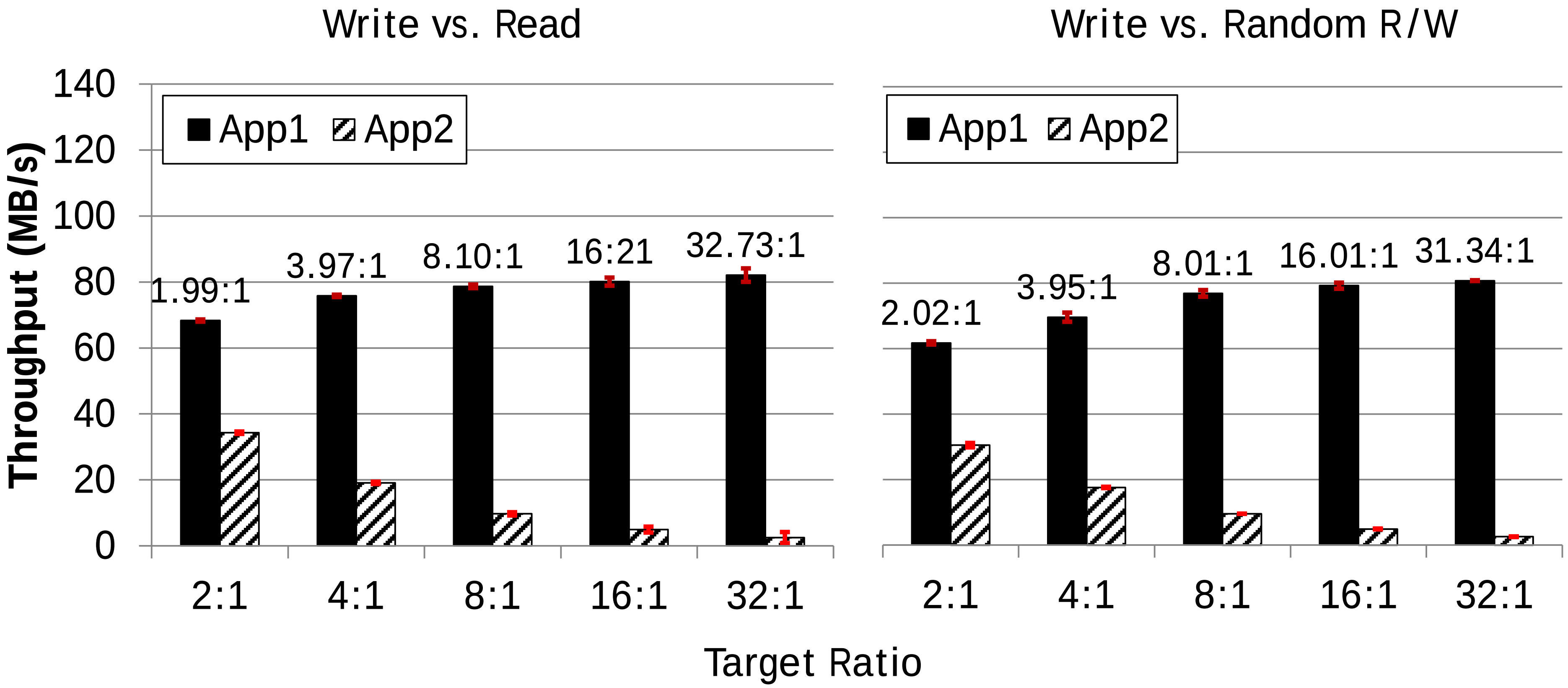


Achieved ratios

Target ratios

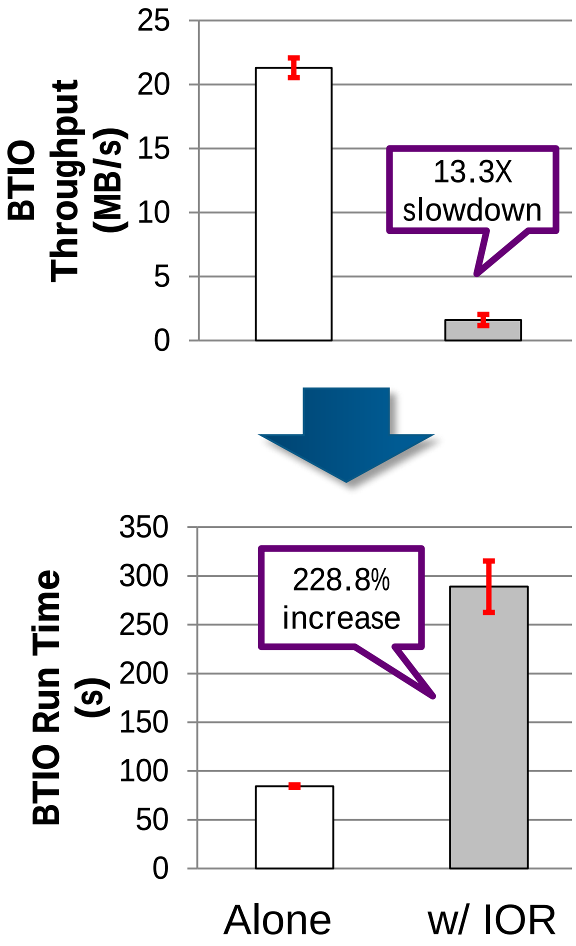
- App1: 4 servers; App2: 8 ervers
- Threshold-driven DSFQ
- 97% accuracy of target sharing ratio is achieved

2 IORs — More Access Patterns



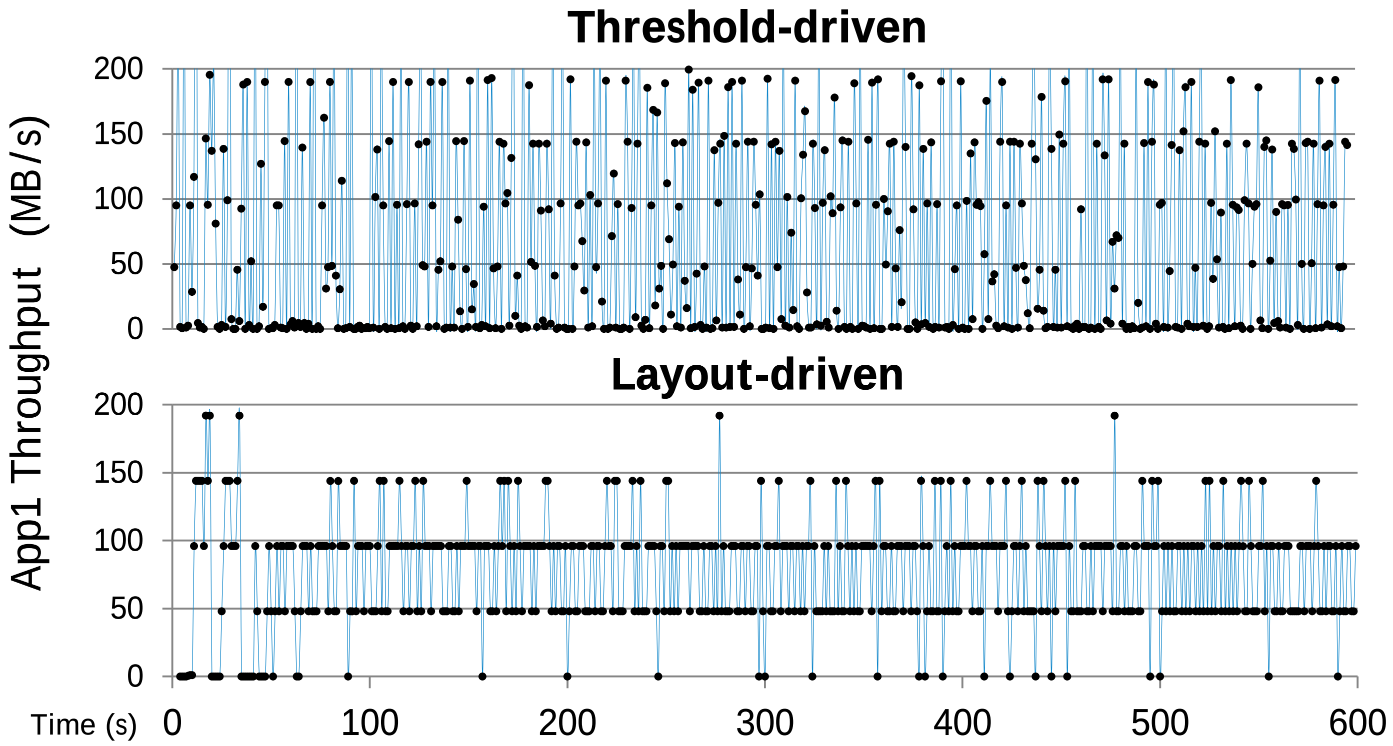
- 97% accuracy of target sharing ratio is also achieved

BTIO vs. IOR



- BTIO & IOR
  - Each with 64 processes
  - 16:1 sharing favoring BTIO
- Layout-driven schedulers
  - Work-conserving
  - Non-work-conserving
- BTIO throughput can be restored to near-standalone performance

Different Synchronization Schemes



- Layout-driven synchronization achieves
  - 13.2% higher throughput
  - 93.0% lower standard deviation

- 8 apps, each with 32 IORs
  - Equal share
- 96 servers
  - Para-virtualized
  - Null-AIO
  - *T* < request size
- Asymmetric file layouts
  - Odd#-app: 48 servers
  - Even#-app: 96 servers

Cost of Implementation

Framework	**LOC**	**Component**	**LOC**
Virtualization	1,692	Interface	694
		TCP	397
		PVFS2	601
Scheduler	2,274	Interface	735
		SFQ(D)	552
		DSFQ	987
Total	3,966		

- The implementation complexity is low for new scheduler / PFS protocol / network support

Conclusions & Future Work

- vPFS manages per-app bandwidth on parallel file system storage by creating virtual PFSes on PVFS2
- vPFS addresses the limitation of distributed algorithms to apply to a parallel storage system
  - Achieves total-service proportional sharing
  - With low-cost synchronization
- Apply the study of QoS-driven parallel storage management on cloud storage
  - Data-intensive
  - Large-scale

References

[1] PVFS2. http://www.pvfs.org/pvfs2/.

[2] PanFS. http://www.panasas.com .

[3] GPFS. http://www.ibm.com/systems/software/gpfs .

[4] Lustre. http://www.lustre.org .

[5] P. Goyal, H. M. Vin, and H. Cheng, “Start Time Fair Queuing: A Scheduling Algorithm For Integrated Services Packet Switching Networks,” IEEE/ACM Trans. Networking, vol. 5, no. 5, 1997.

[6] Yin Wang and Arif Merchant, “Proportional-share scheduling for distributed storage systems,” In Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST’07). USENIX Association, Berkeley, CA, USA, 4-4.

[7] W. Jin, J. S. Chase, and J. Kaur, “Interposed Proportional Sharing For A Storage Service Utility,” SIGMETRICS, 2004.

[8] IOR HPC Benchmark, http://sourceforge.net/projects/ior-sio/.

[9] NASA Parallel Benchmark, http://www.nas.nasa.gov/publications/npb.html .

[10] P. Welsh, P. Bogenschutz, “Weather Research and Forecast (WRF) Model: Precipitation Prognostics from the WRF Model during Recent Tropical Cyclones,” Interdepartmental Hurricane Conference, 2005.

[11] A. Darling, L. Carey, and W. Feng, “The Design, Implementation, and Evaluation of mpiBLAST,” ClusterWorld Conf. and Expo, 2003.

[12] R. Sankaran, et al., “Direct Numerical Simulations of Turbulent Lean Premixed Combustion,” Journal of Physics Conference Series, 2006.

[13] W. Tantisiriroj, et al., “On the Duality of Data-intensive File System Design: Reconciling HDFS and PVFS,” Super Computing, 2011.

[14] MPI-IO, http://www.mpi-forum.org

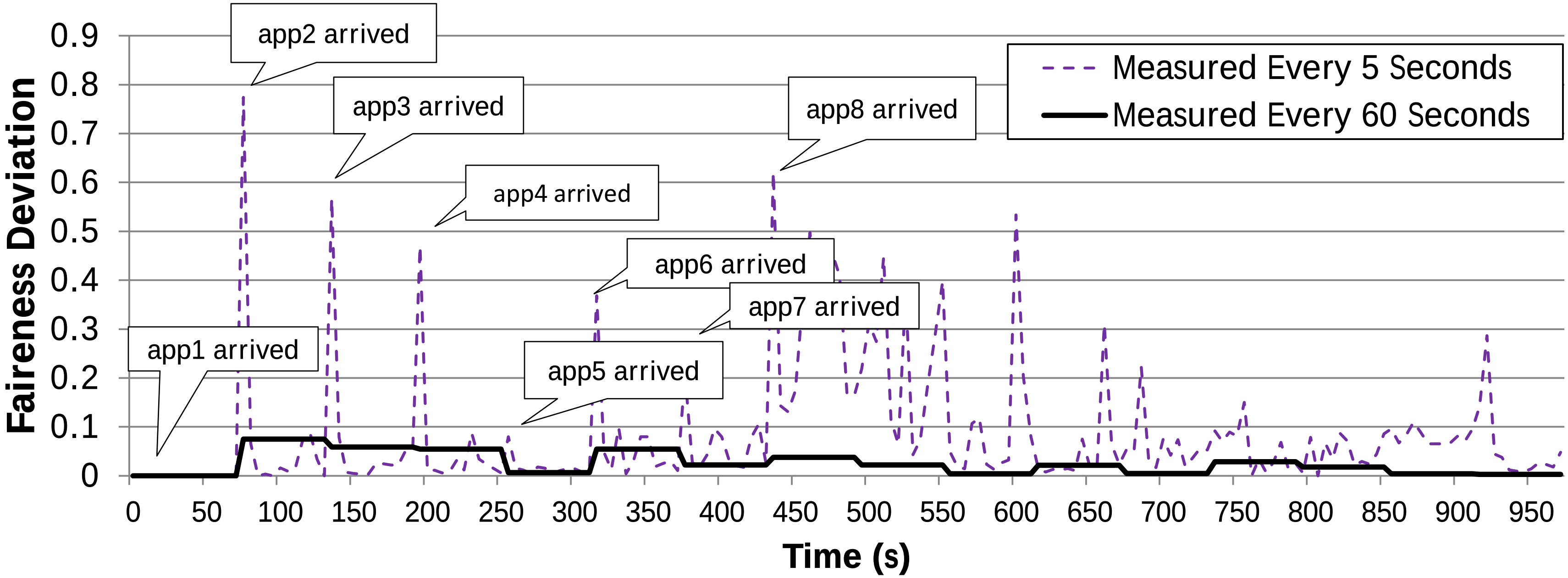
[15] Yiqi Xu, et al., “Technical Report, School of Computing and Information Sciences,” Florida International University http://visa.cis.fiu.edu/tiki/tiki-download_file.php?fileId=51

Acknowledgement

- Research team
  - VISA lab at FIU
    - Yiqi Xu, Dulcardo Arteaga, Dr. Ming Zhao
  - ACIS lab at UF
    - Yonggang Liu, Dr. Renato Figueiredo
  - Industry collaborator
    - Dr. Seetharami Seelam (IBM T.J. Watson Research Center)
- Sponsor: National Science Foundation
- More information: http://visa.cis.fiu.edu/hecura

Backup Slides

8 IORs – Dynamic Arrivals

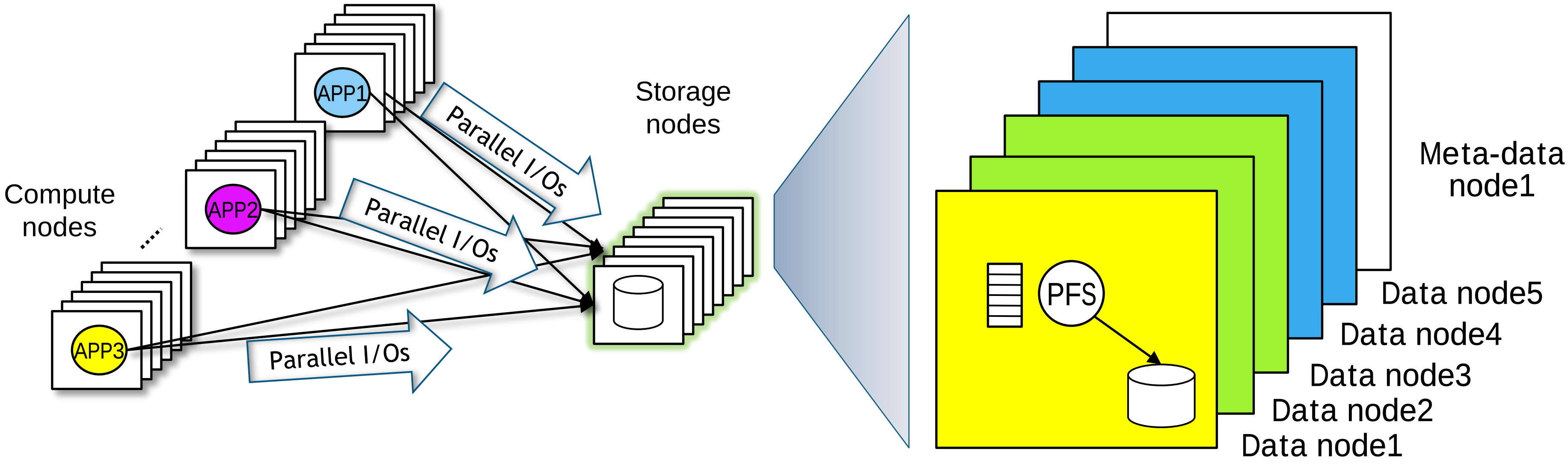


- Unfairness definition: $\sum_{i=1}^{n} |Throughput_i - Weight_i|$
- $Weight_i = i$

Qs

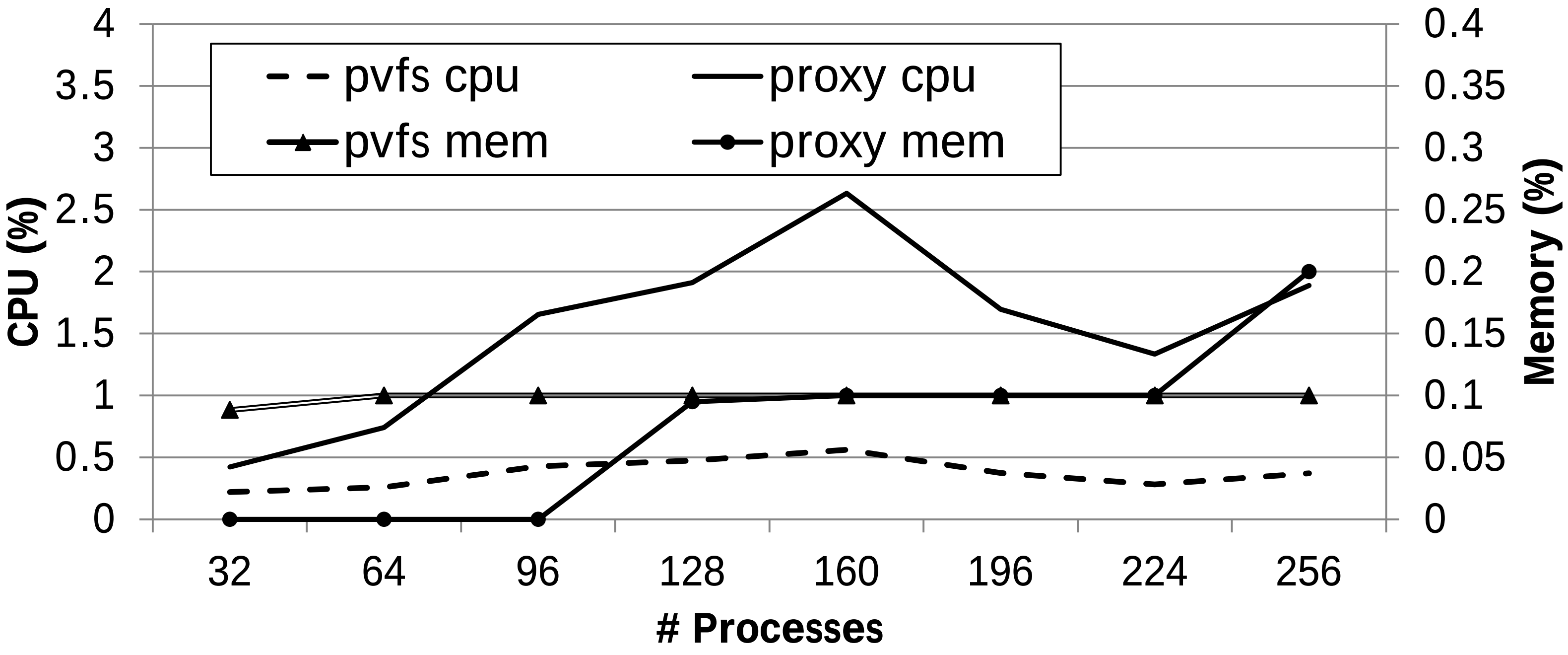
- Fluctuation
- Lower level scheduler affects the higher level

Background



- Parallel File System
  - Distributes data on multiple storage nodes
  - Aggregate throughput from multiple storage nodes
  - File layout — how data is distributed
- Components
  - Server side: data node daemon, meta-data node daemon
  - Client side: MPI library, client daemon

CPU and Memory Overhead

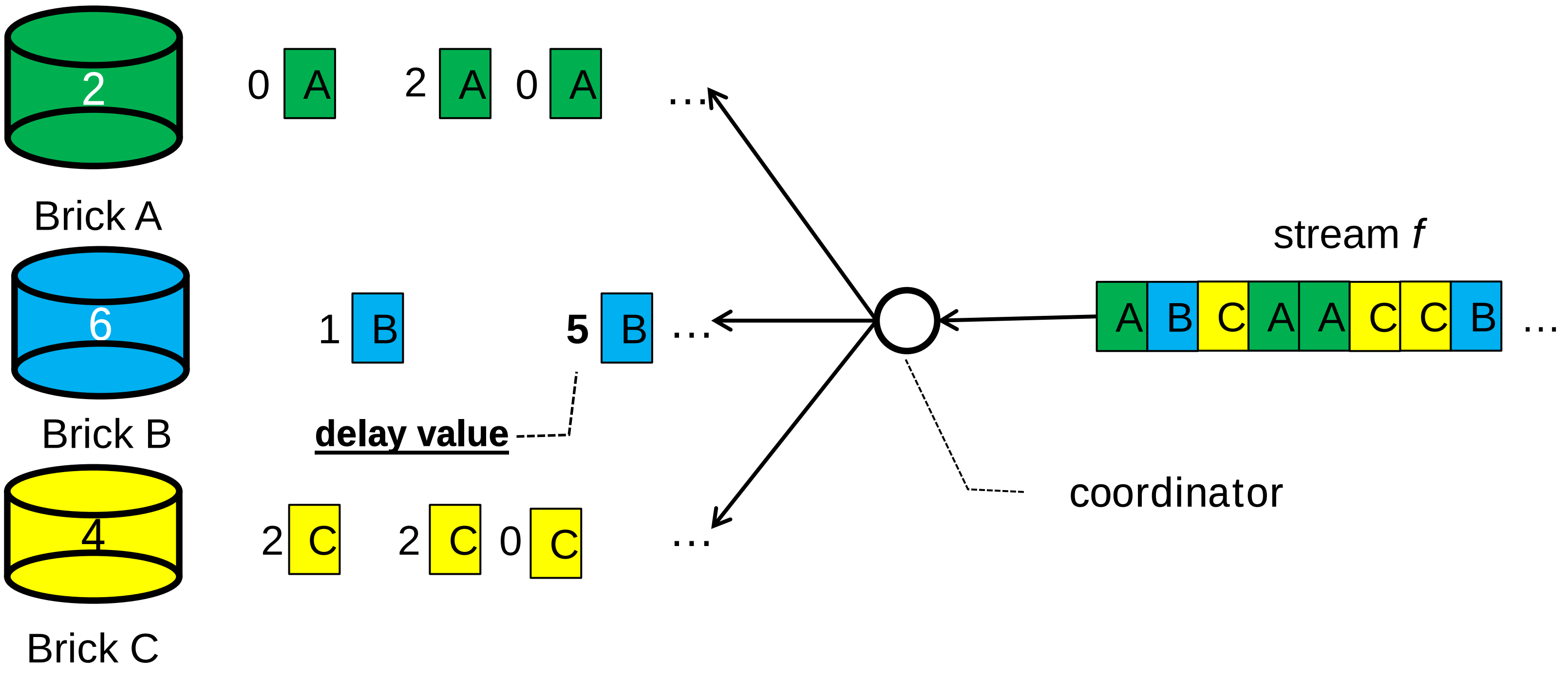


- CPU consumption is below 3%
- Memory consumption is below 0.25%

Difference with Existing Solutions

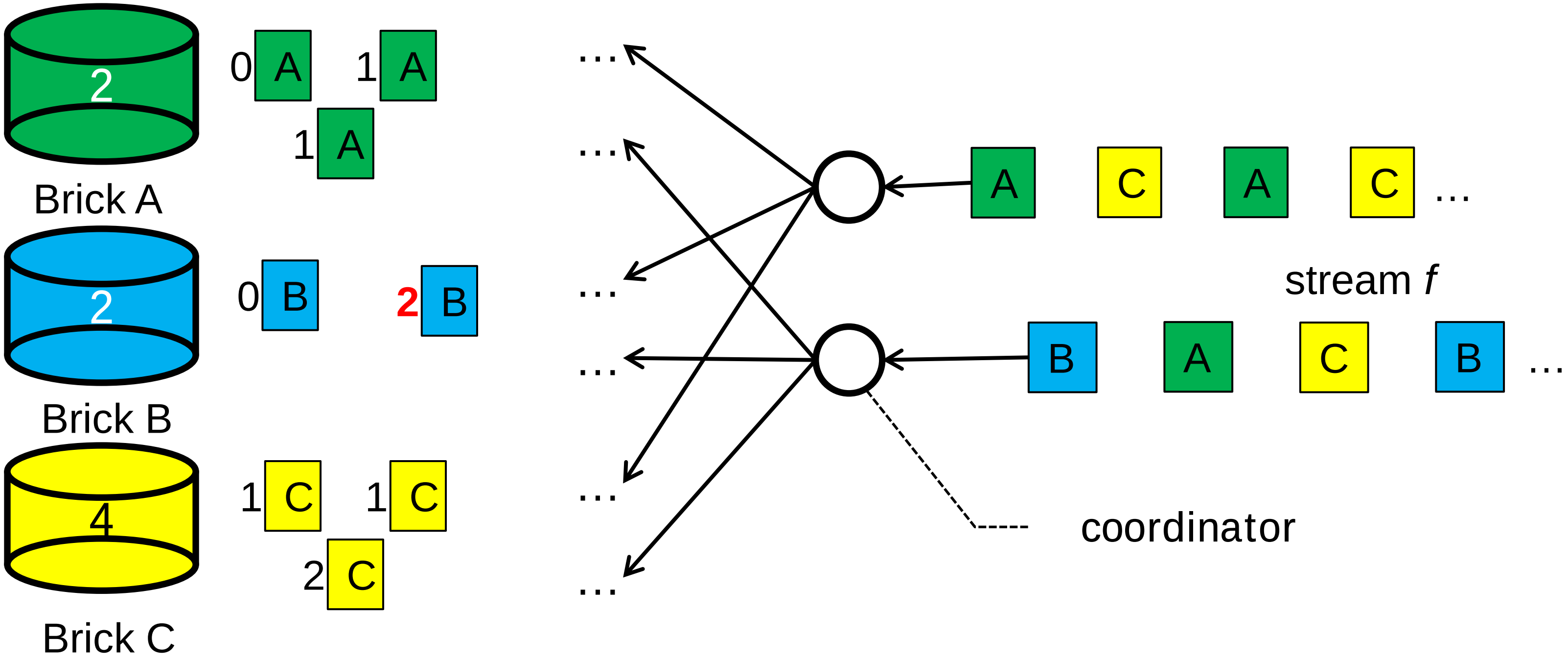
- Facade

Challenges (Single Coordinator)



- Introduces delay value for total-service fair sharing
- Assumption 1: the coordinator can forward I/Os

Challenges (Distributed Coordinators)



- Introduces two or more coordinators
- Assumption 2: clients i.i.d. access to all coordinators