SCALABLE COMPUTING
SOFTWARE LABORATORY

ILLINOIS INSTITUTE OF TECHNOLOGY

IRIS: I/O Redirection via Integrated Storage

Anthony Kougkas, Hariharan Devarajan, Xian-He Sun
akougkas@hawk.iit.edu

Department of Computer Science
Illinois Institute of Technology

ICS'18, Beijing, China
June 12th, 2018

Special thanks to Dr. Shuibing He who kindly accepted to help us present this work.

Agenda

- Background
- Approach
- Design
- Evaluation
- Conclusions
- Q&A

Anthony Kougkas, akougkas@hawk.iit.edu

Highlights of this work

Design of several mapping algorithms
- Files to objects
- Objects to files

Design and implementation of IRIS
- Cross-storage integrated data access
- Programming convenience and efficiency

Evaluation results showed IRIS can offer
- Performance boost up to **7x**
- Minimal overheads

Towards I/O convergence between HPC and HPDA storage subsystems

Different Communities - Cultures - Tools

lustre File System

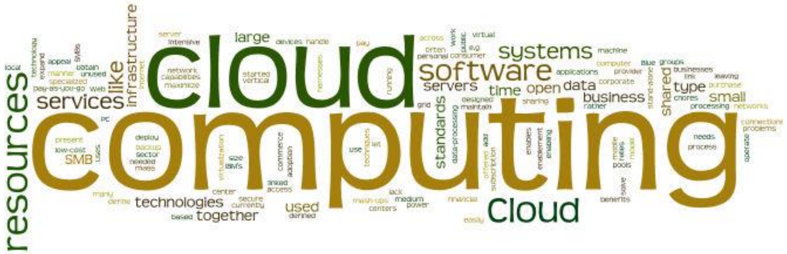
IBM GPFS

OrangeFS

panasas

The tools and cultures of HPC and BigData analytics have diverged, to the detriment of both; **unification** is essential to address a spectrum of major research domains.

- D. Reed & J. Dongarra



cassandra

mongoDB

riak

levelDB

Anthony Kougkas, akougkas@hawk.iit.edu

ILLINOIS INSTITUTE OF TECHNOLOGY

Storage in HPC

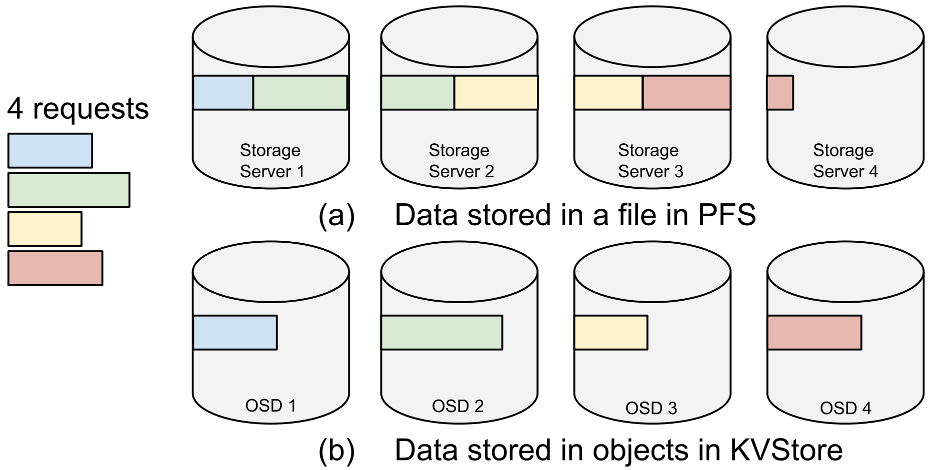
- Parallel File Systems (PFS):
  - Peak performance: ~2000GiB/s
  - Capacity: >70PiB
- Interfaces:
  - POSIX, MPI-IO, HDF5, etc.,
- Limitations:
  - Scalability, complexity, metadata services
  - Small file access, data synchronization, etc.,

#	site.name	site.storage system.net capacity	site.storage system.peak read	site.storage system.peak write	site.storage system.software
		in PiB	*in GiB/s*	*in GiB/s*	
1	kaust	16.96	1955.78	1955.78	Lustre,Cray Tiered Adaptive Storage (TAS)
2	jcahpc	24.10	1918.52	1918.52	Lustre,Lustre
3	riken	39.77	3187.23	1510.85	FEFS,Lustre,FEFS,staging
4	ncsa	27.60	1158.00	1158.00	Lustre,HPSS
5	nscg	14.40	1100.00	1100.00	H2FS
6	llnl	48.85	1000.00	1000.00	Lustre,ZFS
7	ornl	28.00	1000.00	1000.00	Lustre
8	nersc	30.00	700.00	700.00	Lustre,DataWarp
9	jamstec	19.62	407.92	407.92	ScaTeFS,NFS,ScaTeFS,NFS
10	nscc	17.76	288.00	288.00	

I/O 500 List (Nov 2017)

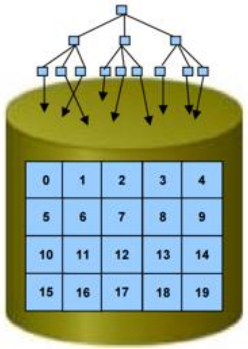
Data Distribution

PFS	KVS
Uses fixed-size stripes	Key-value pair as a single object
Distributes data in a fixed manner(e.g. round robin)	Distributes objects to all available nodes
Need for sub-request synchronization	No need to synchronize anything
Metadata must include the directory tree, permissions, data's physical location on disks, etc.,	Flat namespace with a hash table that keeps the mapping between keys and values



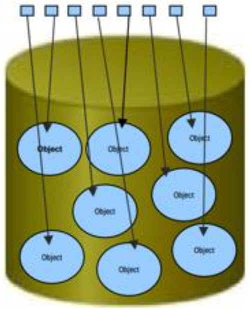
Data models and Storage systems

- File-based storage systems
  - POSIX-I/O
    - fwrite(), fread(),
  - MPI-I/O
    - MPI_File_read(), MPI_File_Write()
  - High-level I/O libraries
    - e.g., HDF5, pNetCDF, MOAB etc



Category	Object Storage	File Storage
Data unit	Objects	Files
Update	Create new object	In-place updates
Protocols	REST and SOAP	NSF with POSIX
Metadata	Custom	Fixed attributes
Strengths	Scalability	Simplified access
Limitations	Frequent updates	Heavy metadata
Performance	High throughput	Streaming of data

- Object-based storage systems
  - REST APIs,
  - get(), put(), delete()
    - Amazon S3,
    - OpenStack Swift
    - NoSQL DBs



- No "one-storage-for-all" solution.
- Each system performs great for certain workloads.
- Unification is essential.

Challenges of I/O Convergence

Gap between	traditional file-based storage	modern scalable data frameworks
Architectural differences	programming models	software tools
Lack of management	heterogeneous data resources	diverse global namespaces

Our Thesis

A radical departure from the existing software stack for both communities is not realistic and	**future software design and architectures will have to raise the abstraction level.**
We aim to design and develop a middleware system which can	**bridge the semantic and architectural gaps.**
We envision a storage system the offers a data path agnostic to the underlying data model and	**that leverages each subsystem's strengths while complementing each other for known limitations.**

Introducing

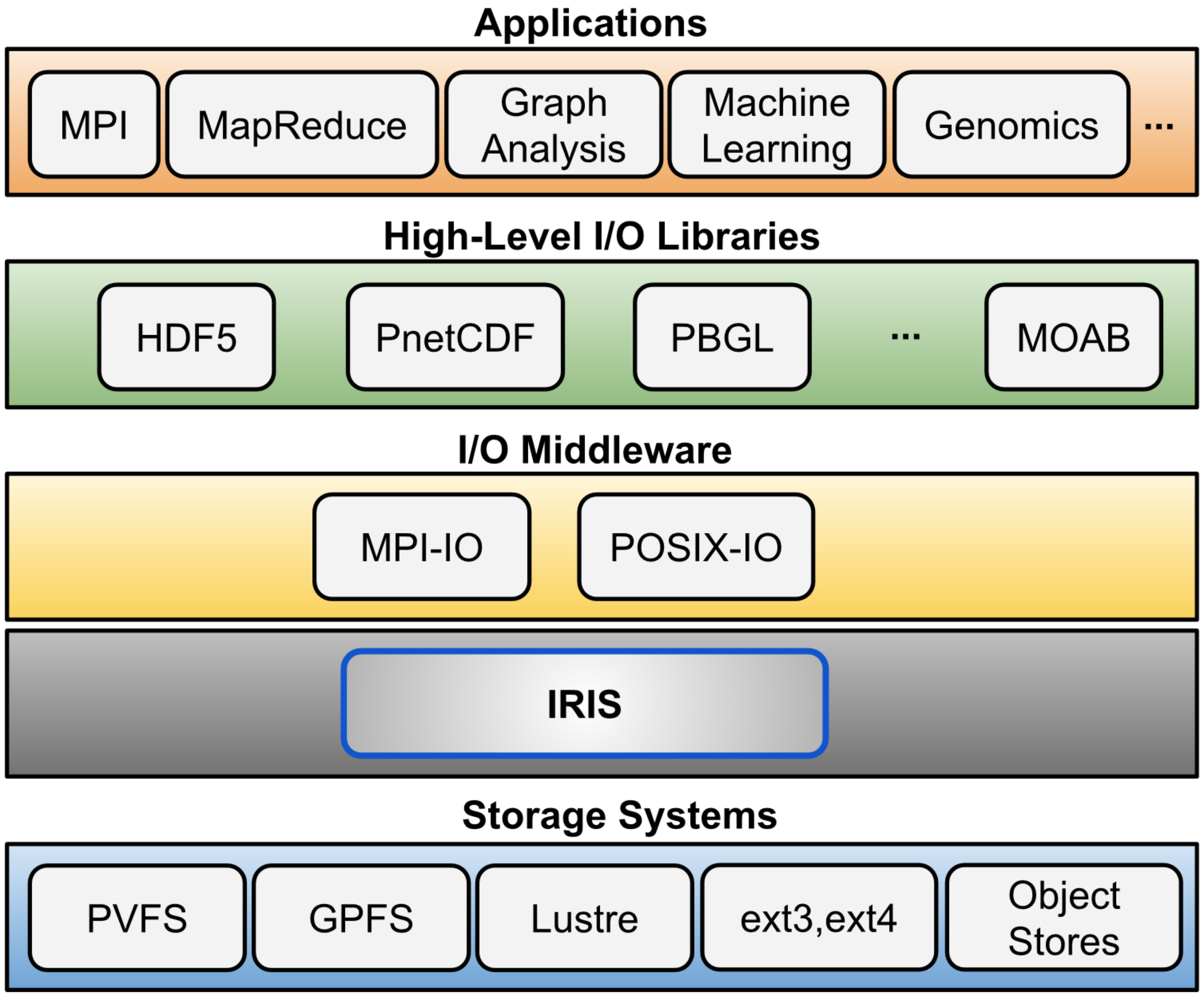
IRIS: I/O Redirection via Integrated Storage

IRIS creates a unified “storage language” to bridge the two very different compute-centric and data-centric storage camps.

Anthony Kougkas, akougkas@hawk.iit.edu

IRIS Design

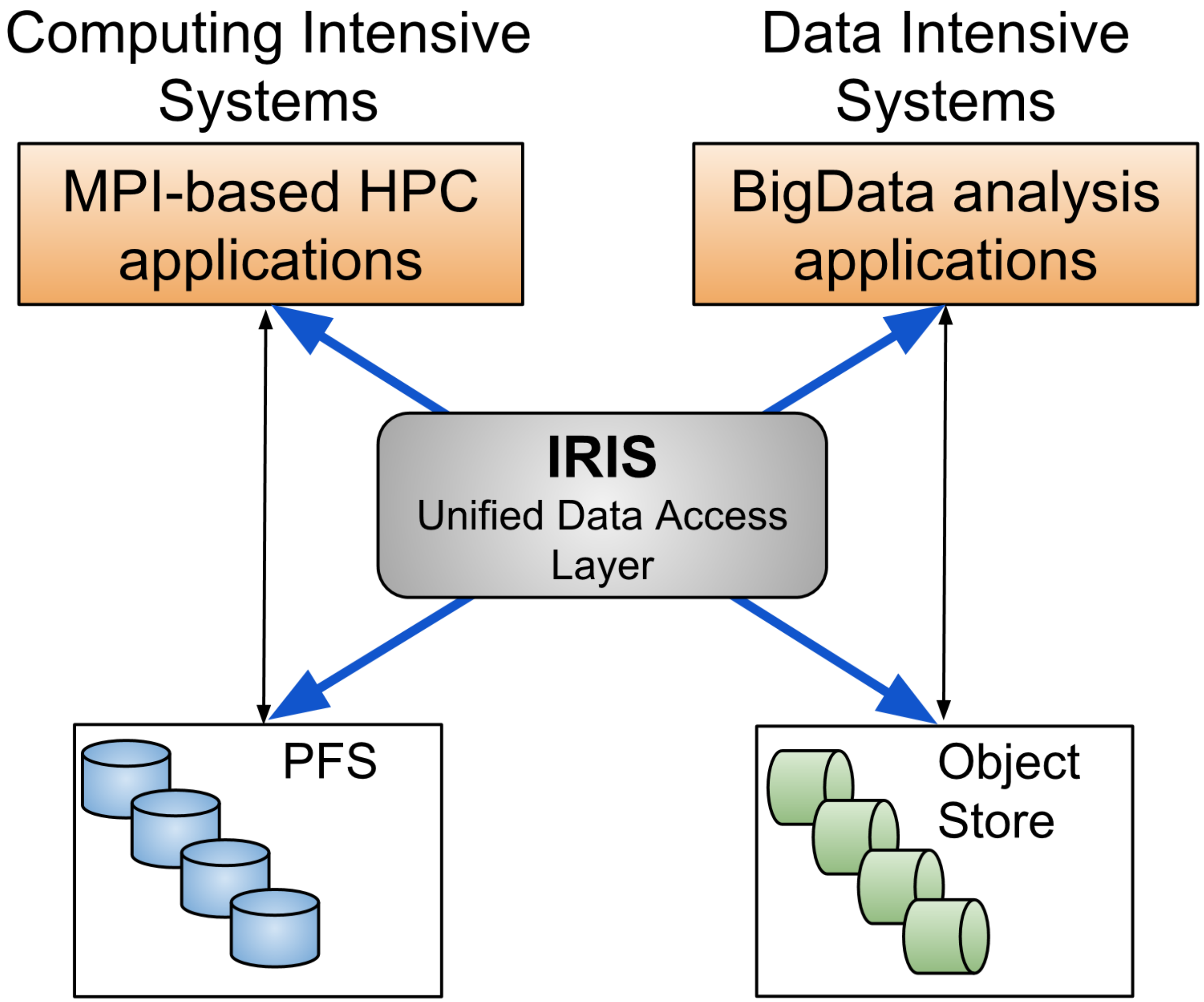
- Middle-ware library
- Wrap-around I/O calls
- Written in C++, modular design
- Non-invasive: plugin nature
  - Link with applications (i.e., re-compile or LD_PRELOAD)
- Existing datasets loaded upon bootstrapping via crawlers
- Directory operations not supported
- Deletions via invalidation



Anthony Kougkas, akougkas@hawk.iit.edu

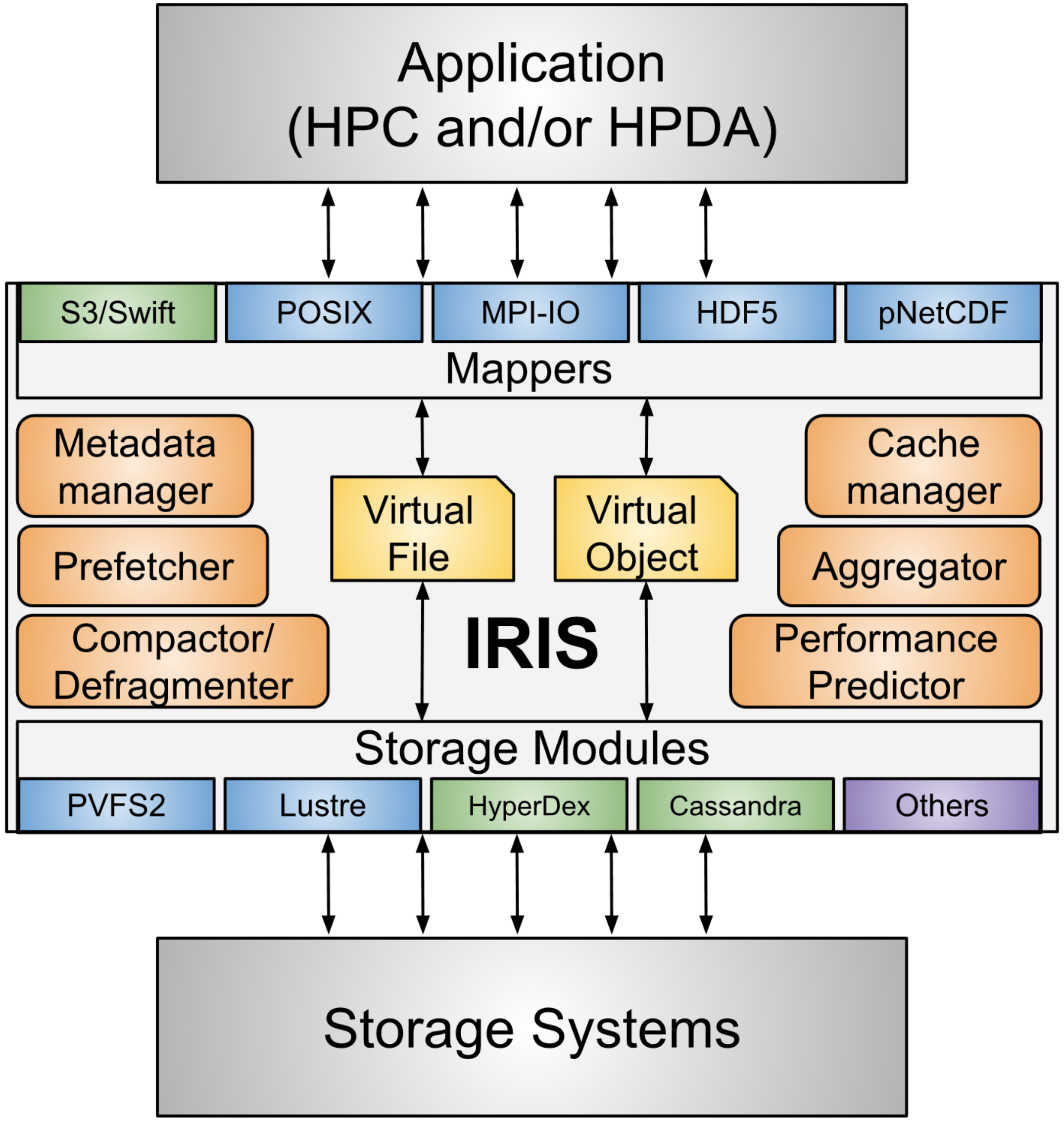
IRIS Objectives

- Enable MPI-based applications to access data in an Object Store without user intervention.
- Enable HPDA-based applications to access data in a PFS without user intervention.
- Enable a hybrid storage access layer agnostic to files or objects.



IRIS Architecture

- Decouples the storage interface
- Abstracts the storage subsystem
- Modular design allows addition of more mappers and modules
- PFS and KVS equal “citizens”
- Optimized for high performance



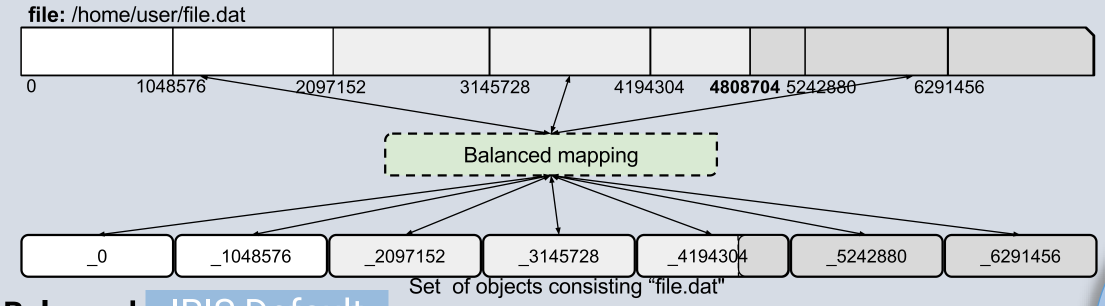
Virtual File (Container)
Name: string
FilePointer: size_t
Size: size_t
Objects: map<VirtualObjects>
InvalidObjects: map<VirtualObjects>

Virtual Object
Name: string
Size: size_t
OffsetInContainer: size_t
Data: void*
LinkedObjects: vector<VirtualObjects>

Mapping Files to Objects

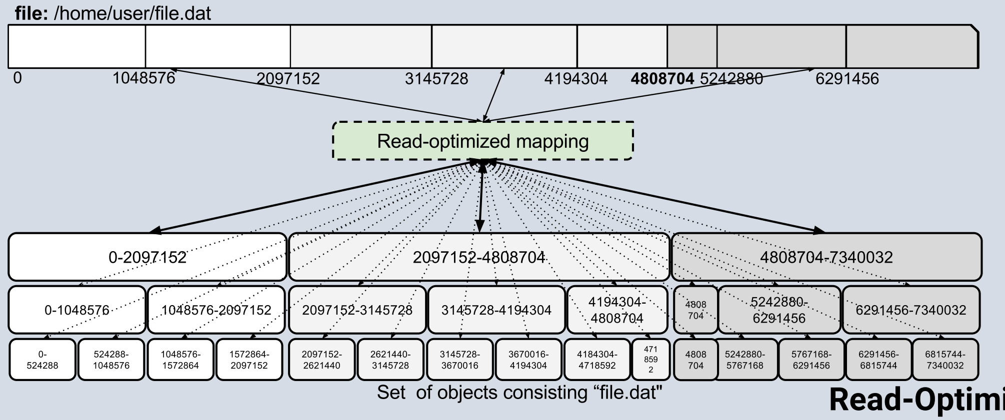
Balanced — IRIS Default

- Ideal for mixed workloads (both fread() and fwrite()).
- File is divided into predefined, fixed-size, smaller units of data, called **buckets**.
- Bucket size is a tunable parameter
- Natural mapping of buckets-to-objects.

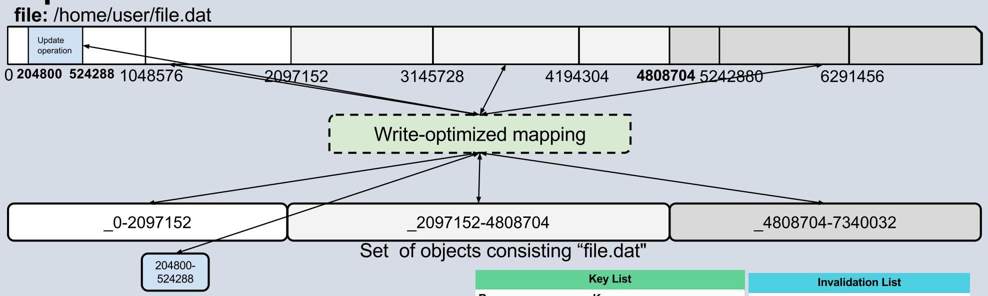


Read-Optimized

- Ideal for read-only or read-heavy (e.g., >90% read) workloads.
- Each write creates a plethora of new various-sized objects.
- Equivalent to replication: sacrifice disk space to increase availability for reads.
- All available keys in a range of offset are kept in a B+ tree.



Write-Optimized

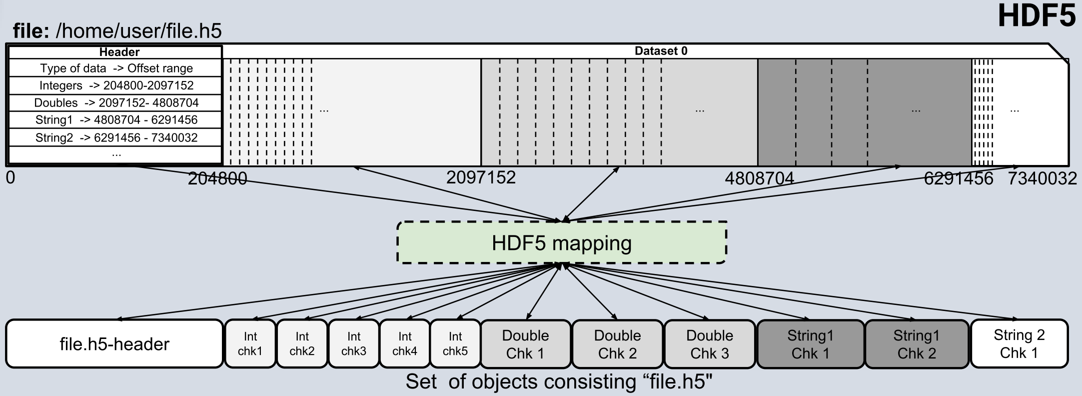


Key List	
Range	Keys
0 - 2097152	file.data_0-2097152
2097152 - 4808704	file.data_2097152-4808704
4808704 - 7340032	file.data_4808704-7340032
0 - 2097152	file.data_204800-524288

Invalidation List	
Key	Range
File.dat_0-2097152	204800 - 524288

- Ideal for write-only or write-heavy (e.g., >80% write) workloads.
- Each request creates a new object.
- A mapping of offset ranges to available keys is kept in a B+ tree for fast searching.
- Update operations create a new object and invalidate ensuring consistency.

HDF5

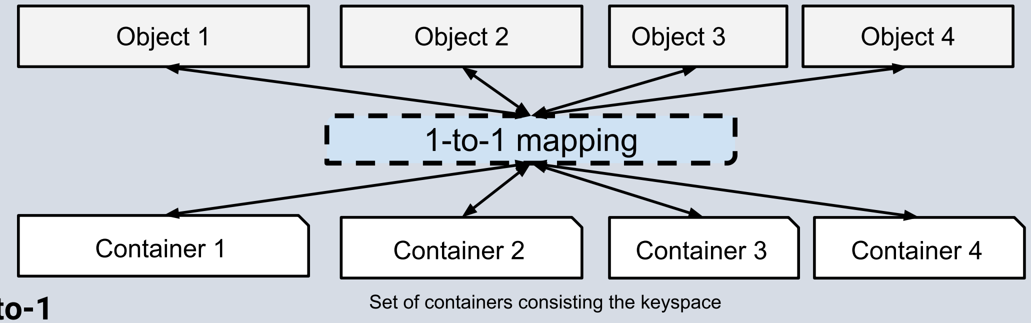


- Exploit rich metadata info HDF5 offers to create better mappings.
- Each HDF5 file creates 2 types of objects: header object and data object.
- Variable-sized data objects are created based on each dataset's dimensions and datatype.

Mapping Objects to Files

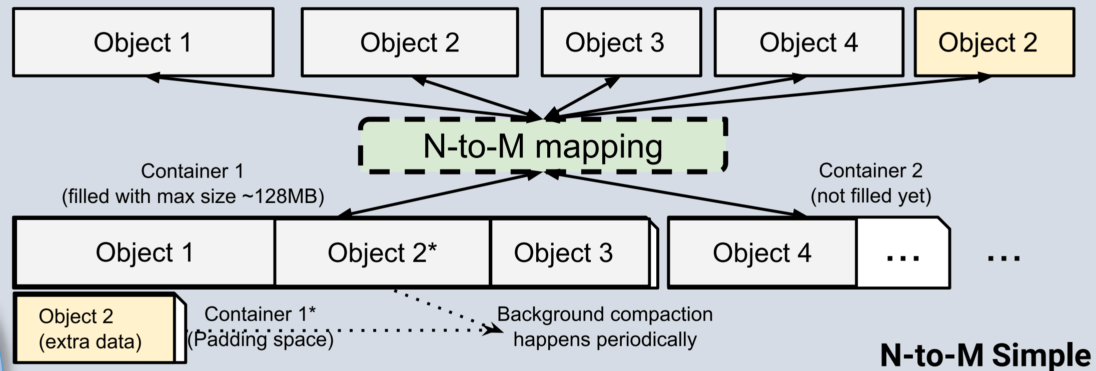
1-to-1

- Each object is mapped to a unique container.
- Ideal for accessing existing datasets.
- Good performance for relatively small number of objects.

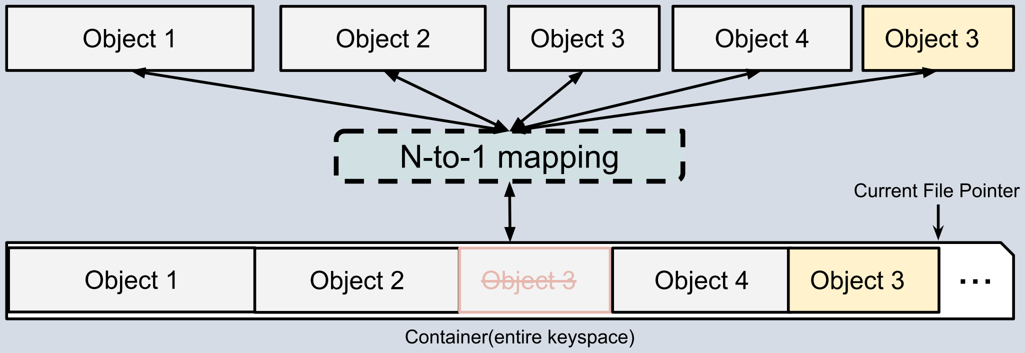


N-to-M Simple

- A collection of objects is mapped to a collection of containers.
- Threshold to create new containers (default every 128MB) bounding the total number of containers.
- Special container-> update container for padding.

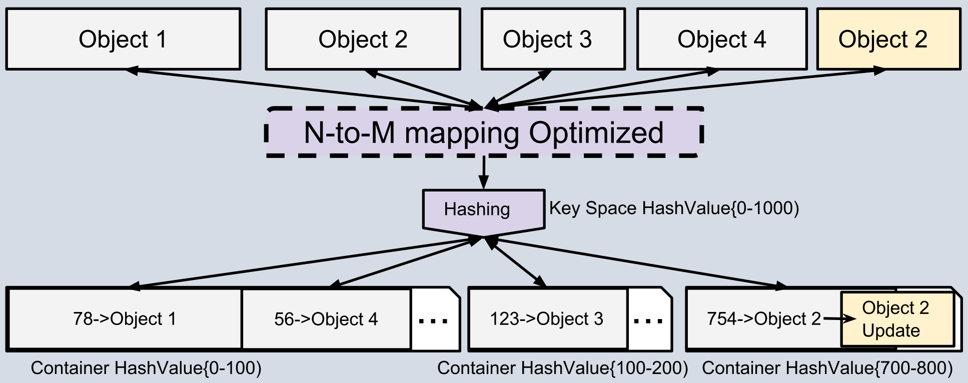


N-to-1

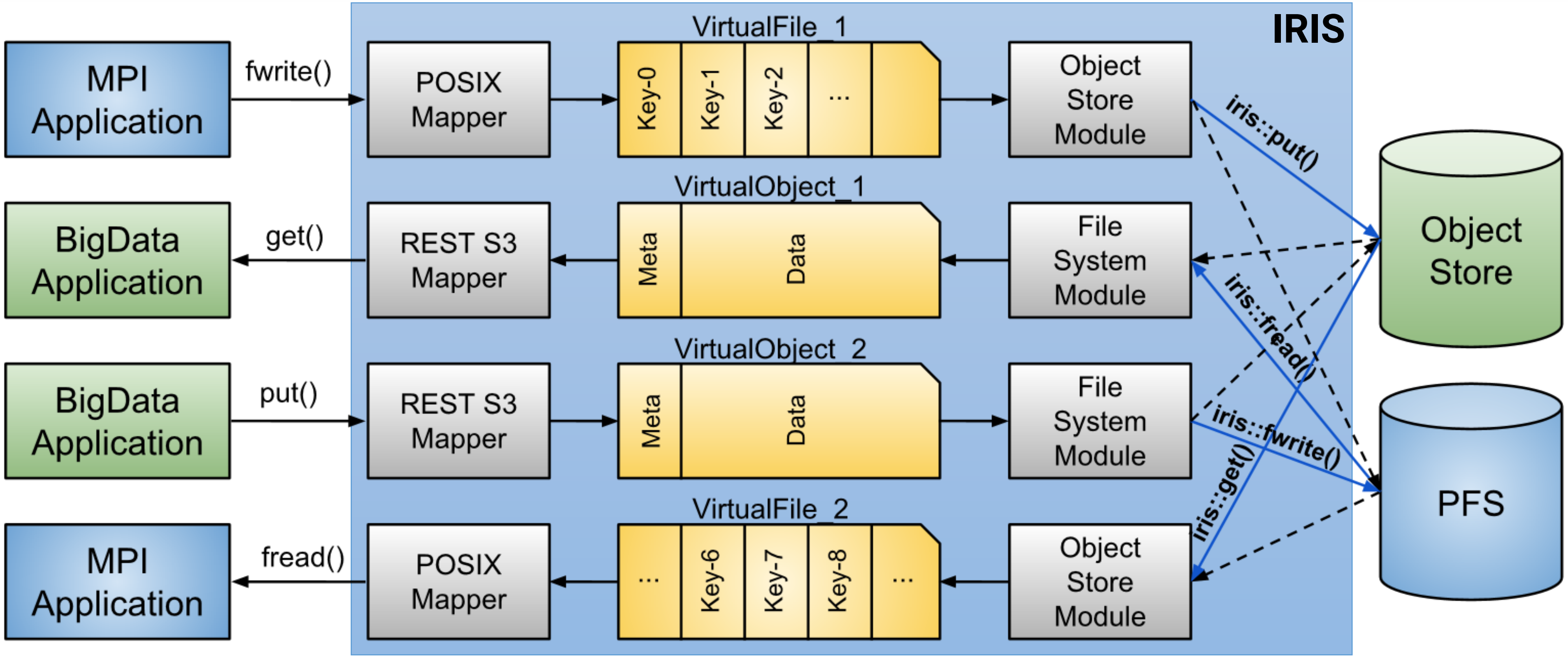


- Entire keyspace is mapped to one container.
- Virtual objects are written sequentially.
- Updates are appended at the end of the file while invalidating the previous object.
- Indexing is important for faster get() operations.

IRIS Default N-to-M Optimized

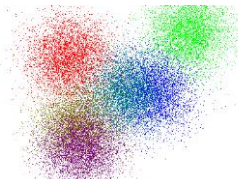


- Objects are first hashed into a key space and then mapped to container
- Containers are created according to a range of hash values and their size is flexible
- Update operations write at the end of the container, invalidating previous object.
- Periodic container defragmentation to save storage space.



Data Flow Example

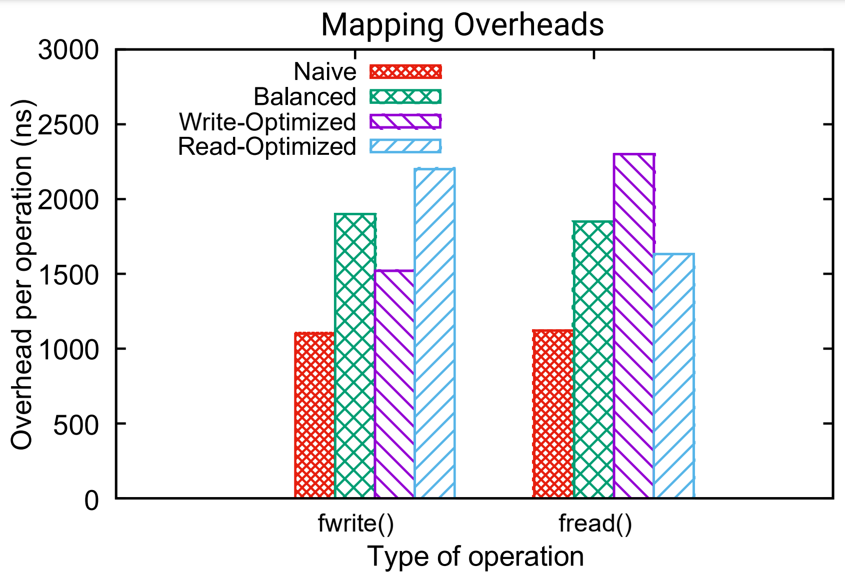
- IRIS enables new data paths
- Abstracts the underlying storage solution



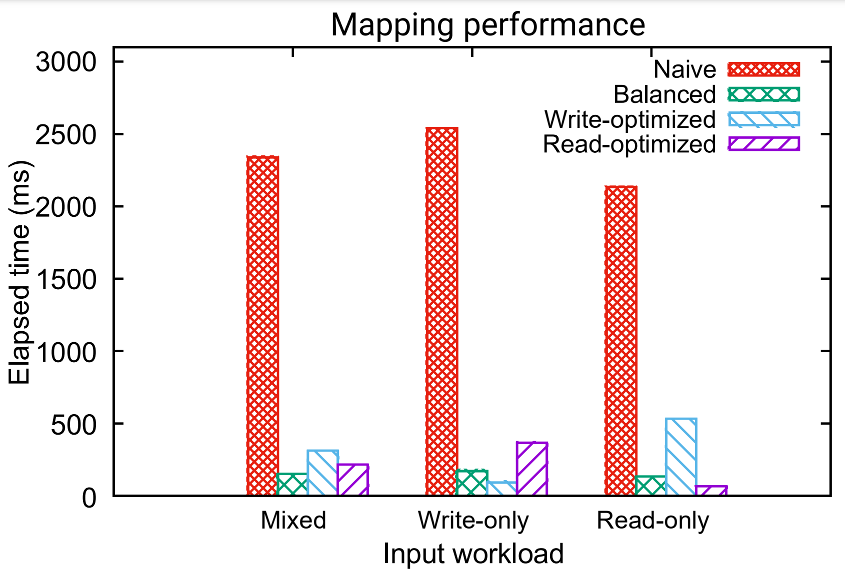
IRIS: I/O Redirection via Integrated Storage
Anthony Kougkas, akougkas@hawk.iit.edu

Testbed and Configurations

- Testbed: Chameleon System
- Appliance: Bare Metal
- OS: Centos 7.1
- Storage:
  - OrangeFS 2.9.6
  - MongoDB 3.4.3
- MPI: Mpich 3.2
- Programs:
  - Synthetic benchmark
  - Montage
  - CM1 from NCSA
  - WRF
  - LAMMPS
  - K-means
  - LANL anonymous scientific simulation



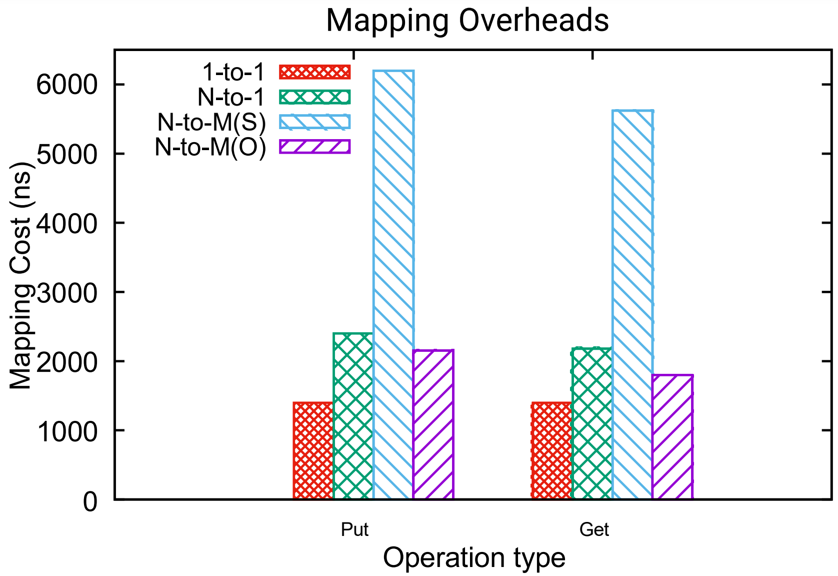
- Input: 65536 POSIX calls
- Output: Average time spend in mapping in ns (per operation)
- Naïve: simple 1-file-to-1-object
- ~0.0050% of the overall execution time (Mapping time over I/O time)



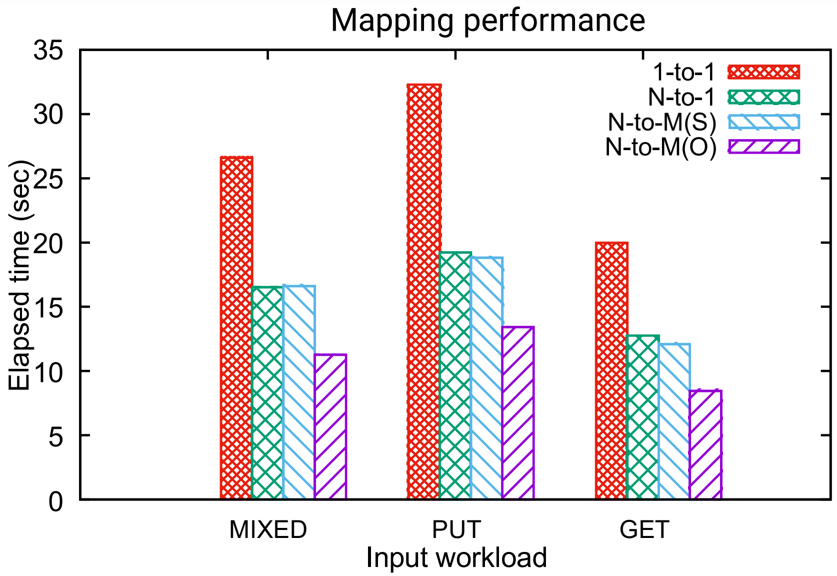
- Request size: 1MB, Total I/O: 32MB, Output: Execution time in ms
- 15x speedup for Balanced and mixed input
- 32x speedup for Read-opt and read-only input
- 27x speedup for Write-opt and write-only input

Mapping evaluation (files-to-objects)

- Overheads are kept to minimum: 2000-3500ns on average or **0.005%**
- Our mapping algorithms outperform the naïve approach by **15-32x**



- Input: 128K objects of 64KB
- Output: Average time spend in mapping in ns (per operation)
- 1-to-1 simplest with lower cost
- N-to-M-Optimized only 2000ns

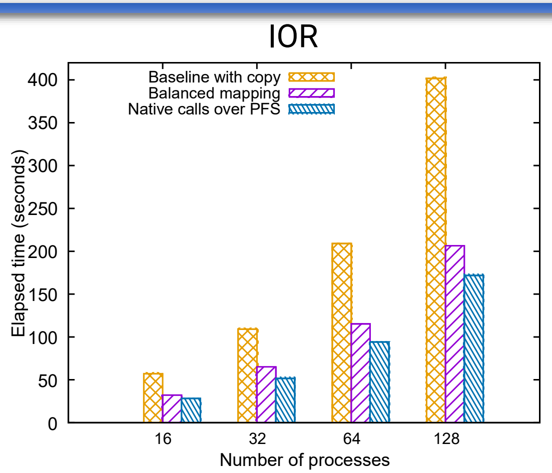


- Workload: 4GB total I/O, Object size: 64KB, Output: Overall time in seconds
- Flow for mixed: 1GB write, 1GB read, 1GB update followed by 1GB read
- 1-to-1 suffers from large number of files
- N-to-M-Optimized performed more than 2x faster

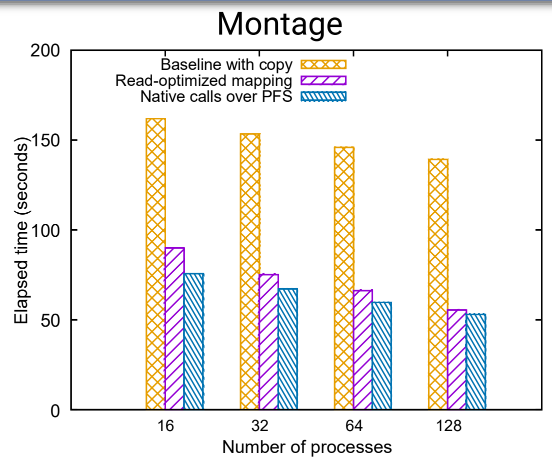
Mapping evaluation (objects-to-files)

- Overheads are kept to minimum: 1300-6000ns on average or **0.008%**
- Our mapping algorithms outperform the naïve approach by **15-32x**

Anthony Kougkas, akougkas@hawk.iit.edu

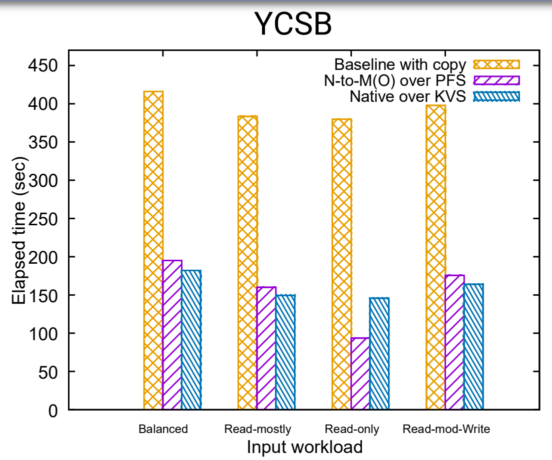


- 4 client - 4 servers
- MPI-IO, Blk_size=2MB, Transfer size = 512KB, Total I/O = 512MB per process, File-per-process, DirectIO
- Output: Execution time in seconds
- Baseline: first copy the input files from MongoDB to OrangeFS and then run

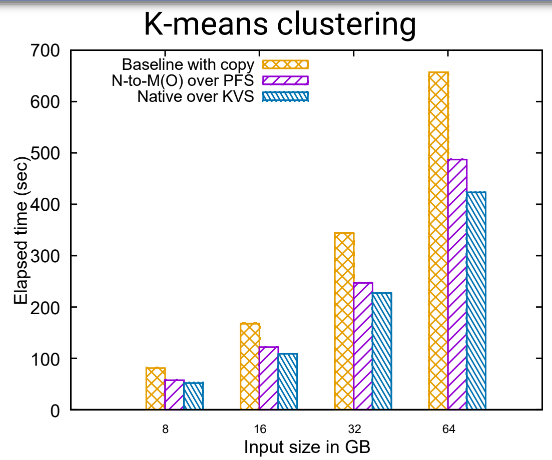


- 4 client - 4 servers
- POSIX-I/O, Total I/O = 24GB, File-per-process, OS cache disabled and flushed before
- Output: Execution time in seconds
- Baseline: first copy the input files from MongoDB to OrangeFS and then run

Files-to-Objects



- Workloads:
  - Balanced: 50% reads and 50% writes
  - Read-mostly: 90% read and 10% writes
  - Read-only: 100% read
  - Read-modify-write
- Total I/O 64GB in 64KB objects
  - Data are copied into the Redis and then run the test natively



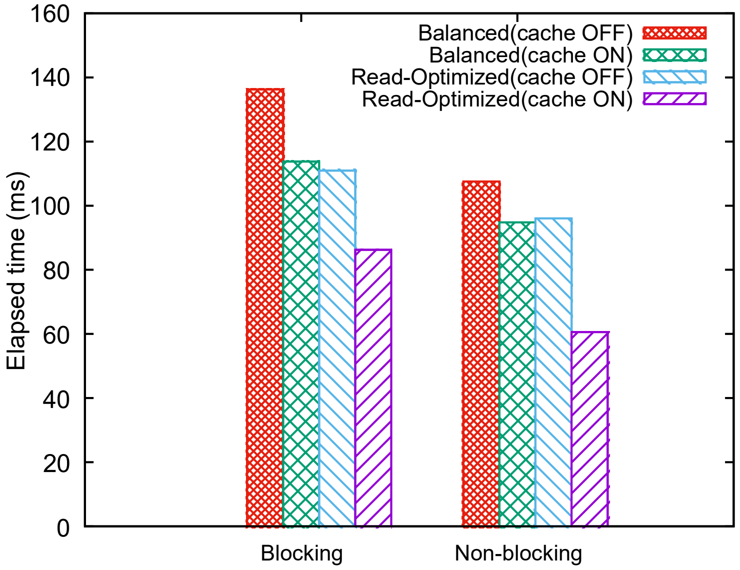
- Offline data preloading
- Baseline flow:
  - Data are copied into the Redis and then run natively
- Minimal overhead
  - 57 sec over 52 sec natively for 8GB

Objects-to-Files

Mapping Performance (real workloads)

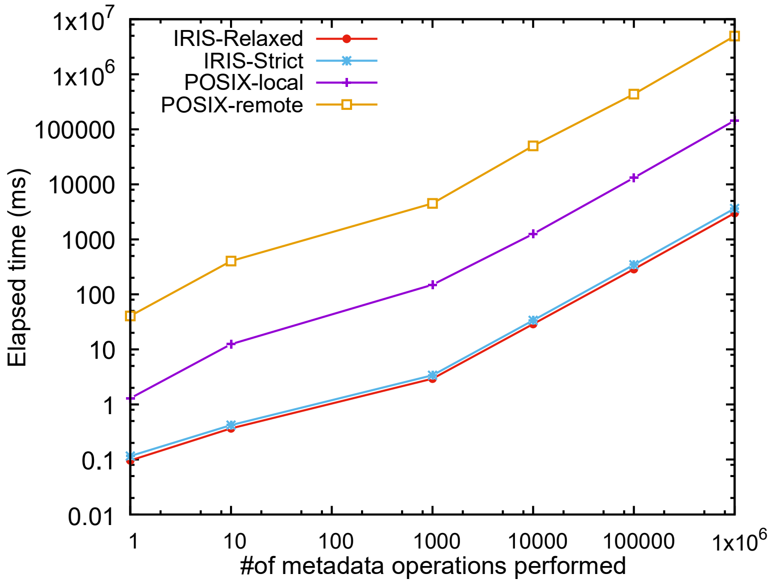
With careful mapping between the two data formats,
IRIS demonstrates **more than 2x speedup**

Prefetcher



- Blocking -> synchronous, Non-blocking-> asynchronous
- Cache ON/OFF reflects caching previous write operations
- A combination of caching and non-blocking shows **2x performance gain**

MDM (POSIX)



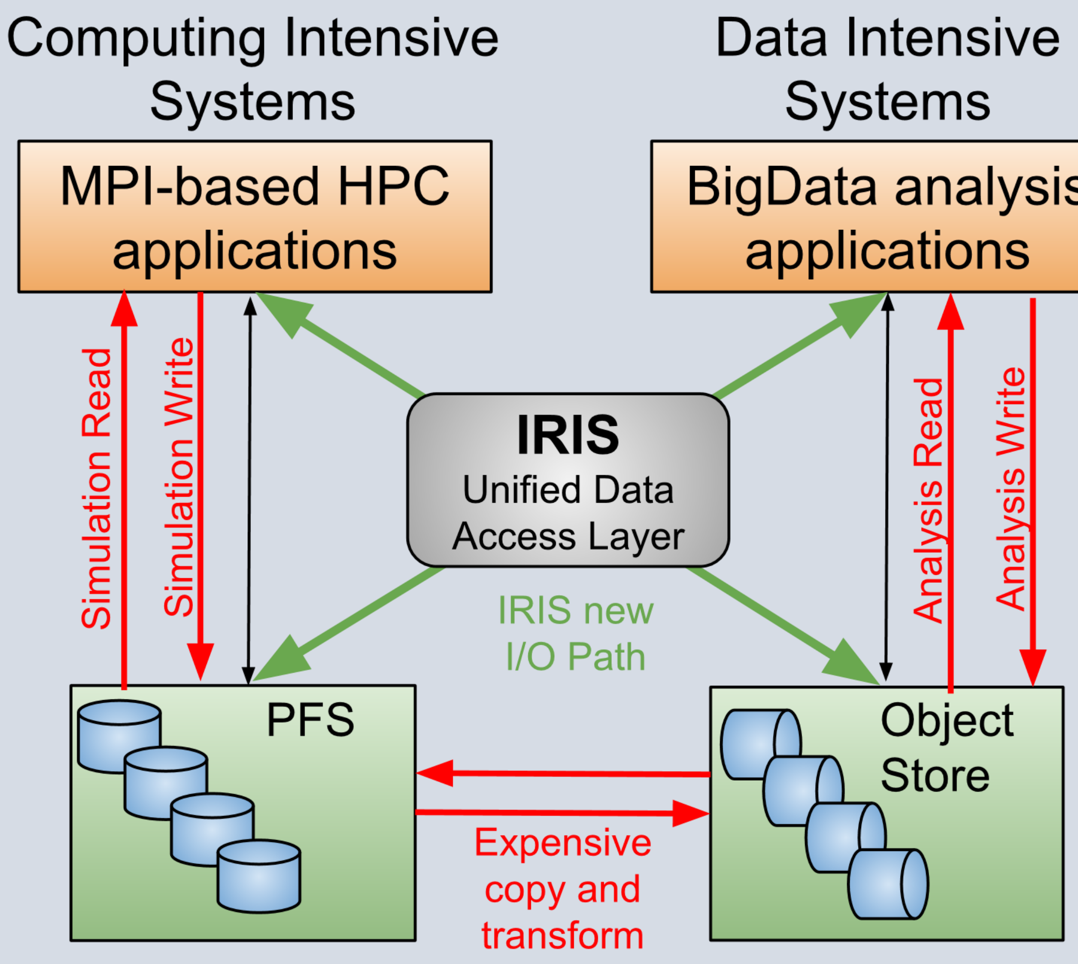
- IRIS strict complies with POSIX standard for maximum compatibility
- IRIS relaxed only maintains metadata relevant to the object mapping
- Relaxed mode offers **18% boost**

IRIS Components Evaluation

- Prefetcher can speed read operations up to **2x**
- POSIX **compliant** in-memory MDM

Workflow Evaluation

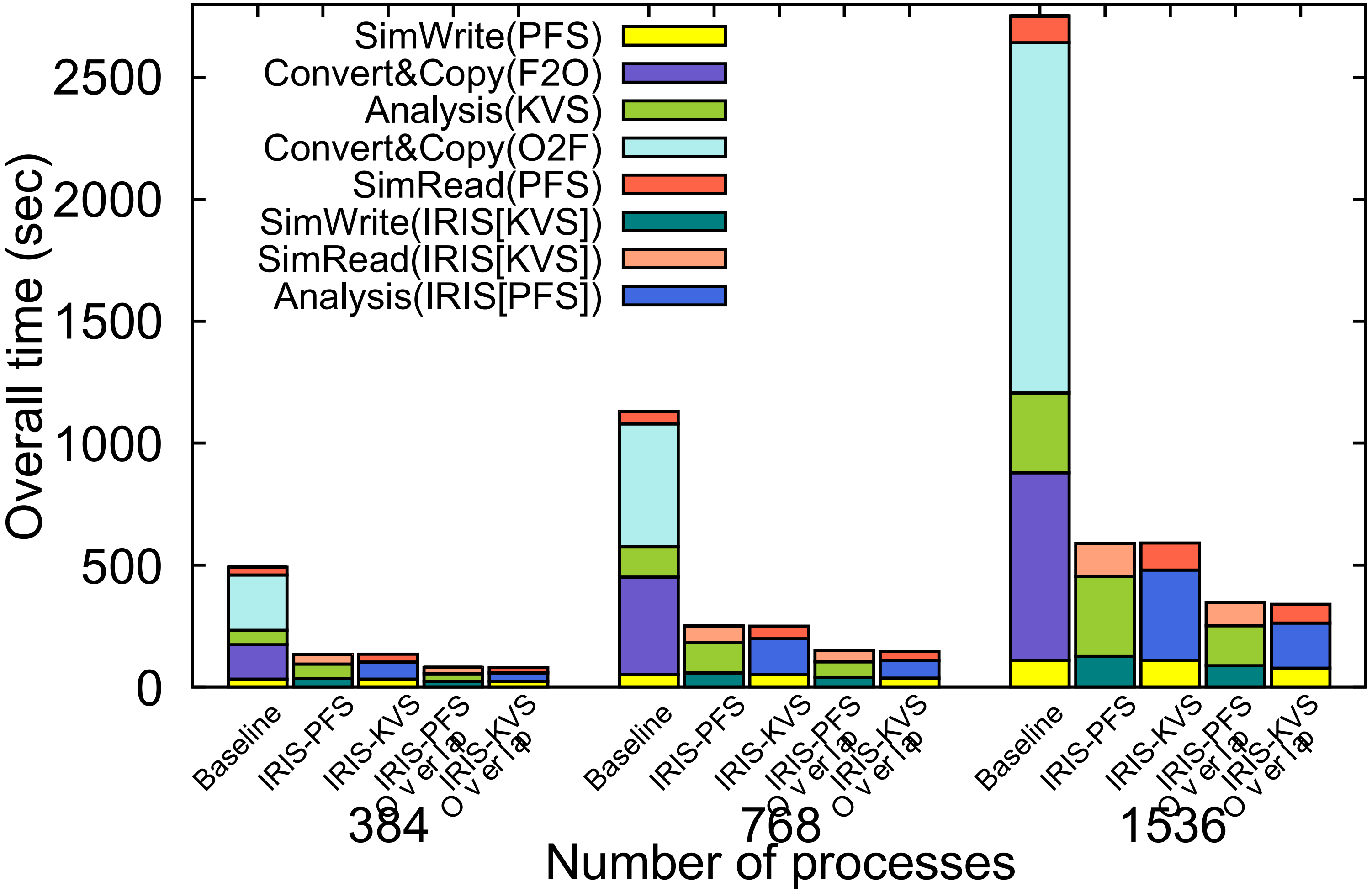
- F2O: copying from PFS to KVS
- O2F: copying from KVS to PFS
- IRIS[PFS]: IRIS operations on top of PFS
- IRIS[KVS]: IRIS operations on top of KVS
- Total time is a compound of several phases.



$$\boldsymbol{Total\ Time} = Simulation\ Write + \text{Copy Data PFS2KVS} + \text{Analysis Read} + \text{Analysis Write} + \text{Copy Data KVS2PFS} + \text{Simulation Read}$$

Real Applications

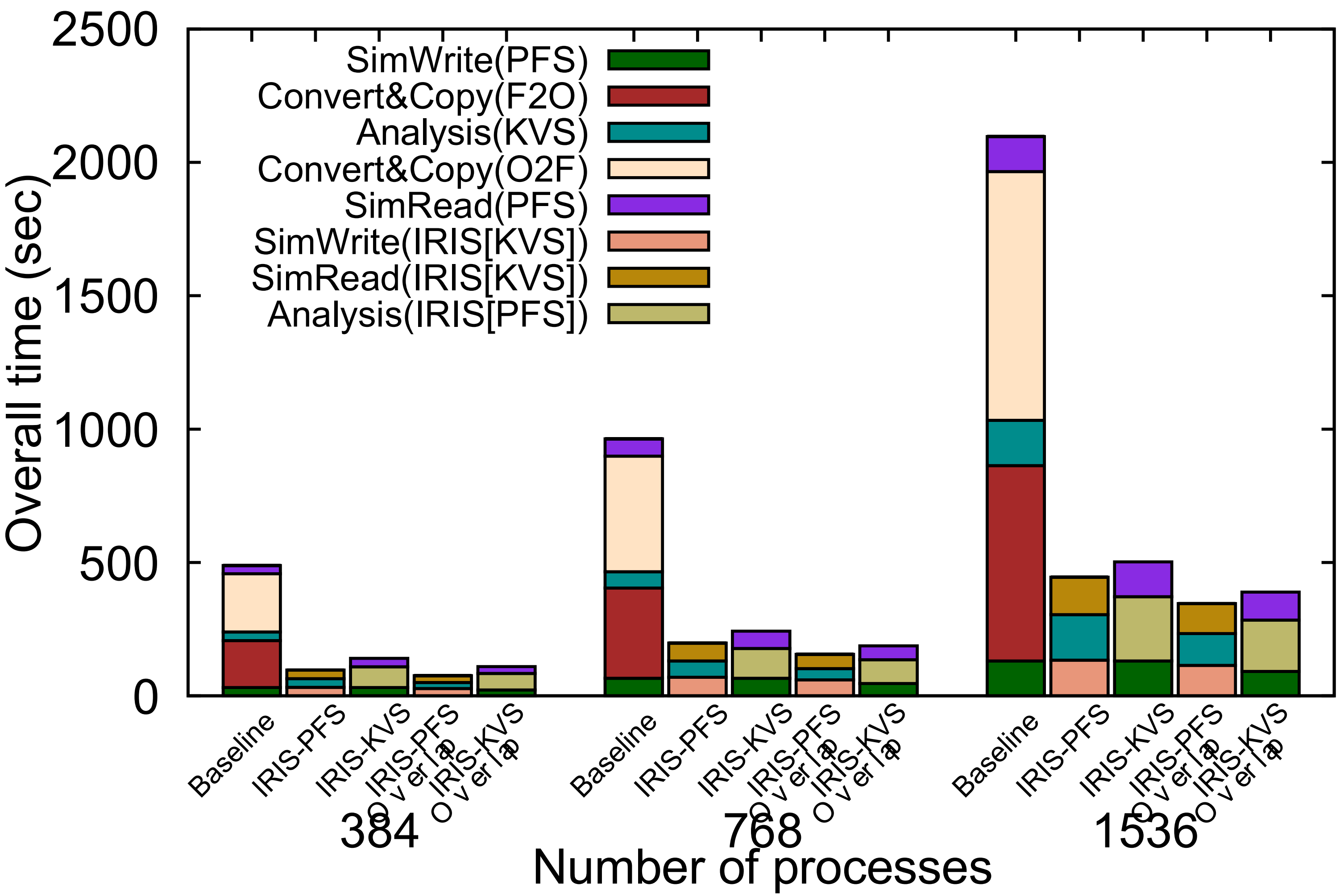
- File-per-process
- Each rank 100MB (150GB total)
- **5x improvement** over baseline
- **7x improvement** overlap mode
- Analysis is 1.7x slower when run on top of PFS
- Copying dominates total time



CM1

Real Applications

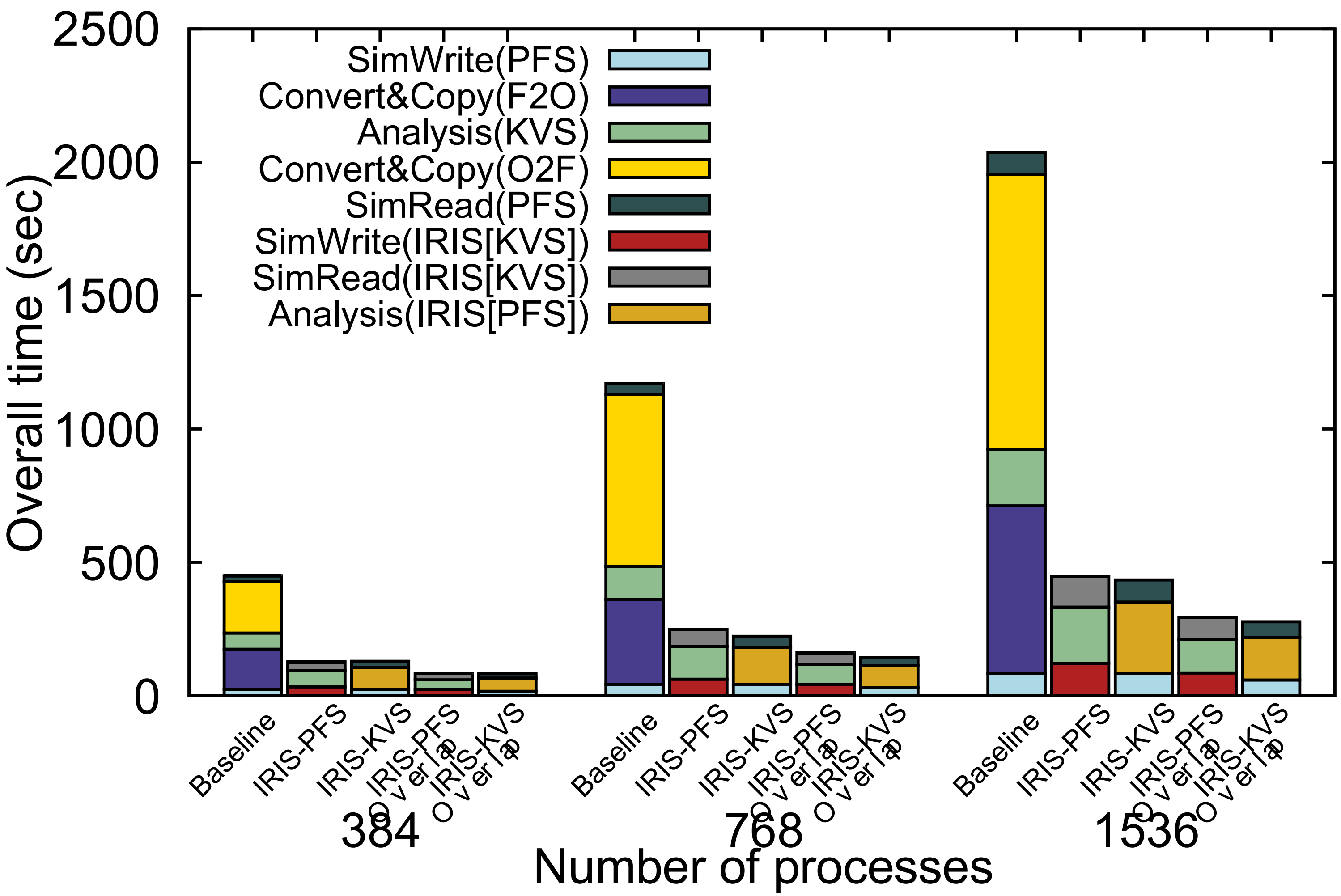
- File-per-process
- Each rank 100MB (150GB total)
- **4x improvement** over baseline
- **6x improvement** overlap mode
- Analysis is 2.2x slower when run on top of PFS



Montage

Real Applications

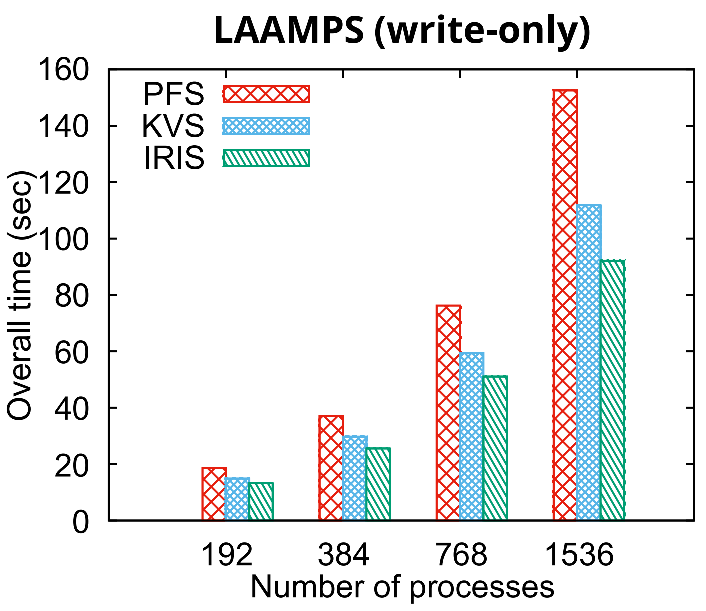
- File-per-process
- Each rank 100MB (150GB total)
- **5x improvement** over baseline
- **7x improvement** overlap mode
- Copying dominates total time
- Analysis is 50% slower when run on top of PFS

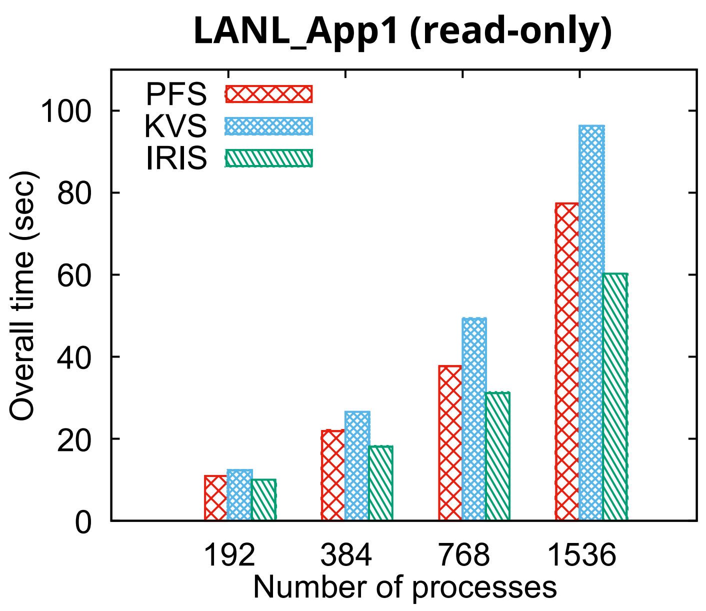


WRF

IRIS in hybrid mode

- Large requests placed in PFS
- Small access directed to KVS
- Each process 32MB total 48GB
- LAMMPS:
  - Threshold: 64KB
  - Ratio small/large requests: 2-to-1
  - Favoring KVS
- LANLApp1:
  - Threshold: 128KB
  - Ratio small/large requests: 1-to-4
  - Favoring PFS





65% improvement over PFS
21% improvement over KVS

28% improvement over PFS
59% improvement over KVS

Related Work

- From the File system side:
  - CephFS
  - PanasasFS
  - OBFS: A File System for Object-based Storage Devices OSD
- From the Object store side:
  - AWS Storage Gateway
  - Azure Files and Azure Disks
  - Google Cloud Storage FUSE

IRIS is a general solution that can bridge any File System with any Object Store and does NOT require change in user code or underlying system deployments.

In summary

- HPC and HPDA infrastructures are converging.
- File-based and Object-based storage solutions:
  - We need to raise the level of abstraction.
- IRIS implements several new algorithms to map files to objects and vice versa.
- IRIS offers:
  - Programming convenience
  - Legacy application support
  - Transparent cross-storage integrated data access
  - Performance improvements up to **7x**

IRIS: I/O Redirection via Integrated Storage

Thank you.

Anthony Kougkas

akougkas@hawk.iit.edu

https://sites.google.com/iit.edu/akougkas

National Science Foundation

WHERE DISCOVERIES BEGIN

This work was supported by the National Science Foundation under grants no. CCF-1744317, CNS-1526887, and CNS-0751200.

SCALABLE COMPUTING SOFTWARE LABORATORY

ILLINOIS INSTITUTE OF TECHNOLOGY