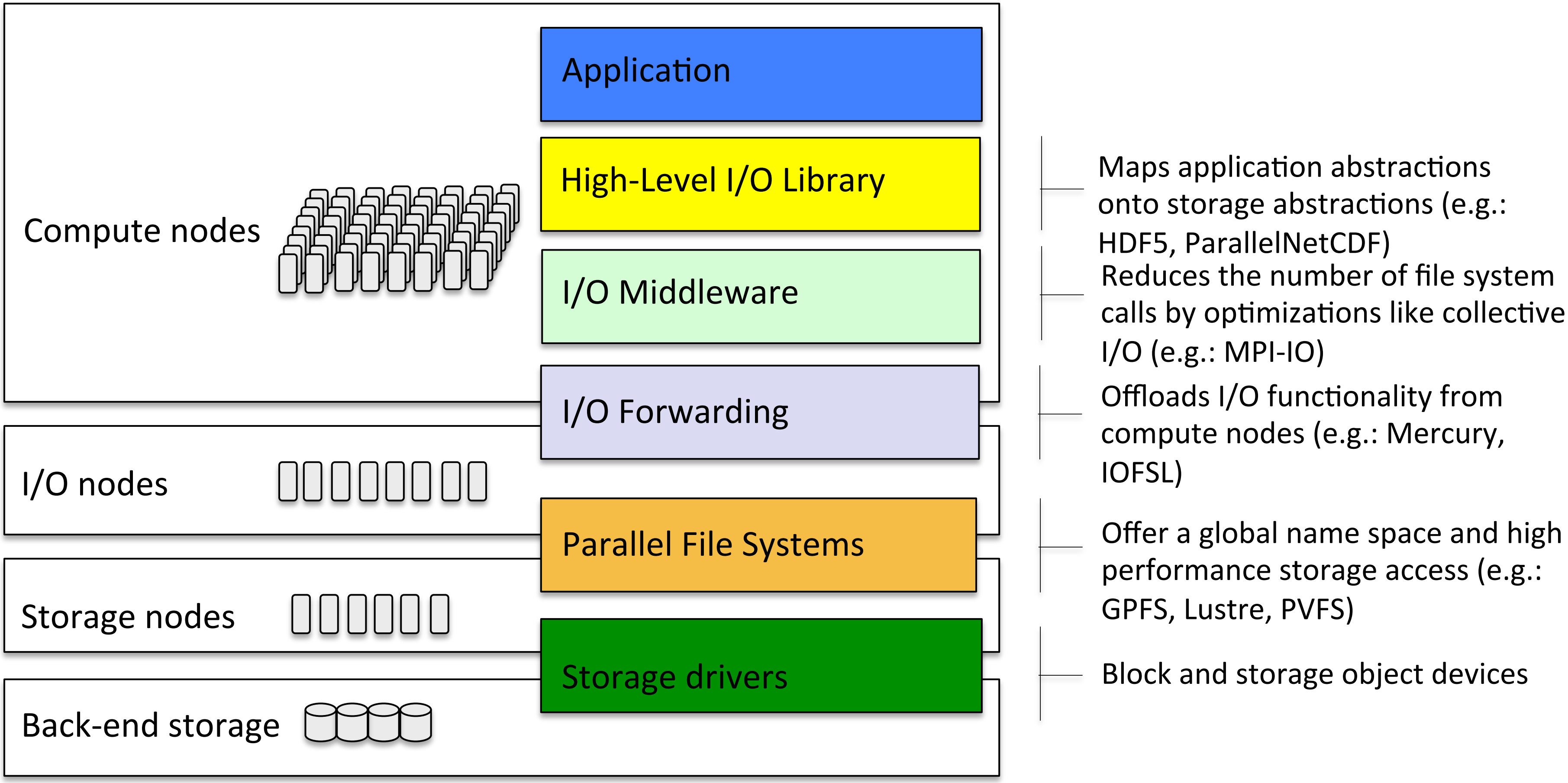
Optimizing data staging based on autotuning, coordination, and locality exploitation on large scale supercomputers

Florin Isaila

ANL & University Carlos III

Collaborators: Prasanna Balaprakash (ANL), Phil Carns (ANL), Jesus Carretero (UC3M), Franscisco Duro (UC3M), Javier Garcia (UC3M), Kevin Harms (ANL), Paul Hoveland (ANL), Dries Kimpe (ANL), Rob Latham(ANL), Tom Peterka (ANL), Rob Ross (ANL), Stefan Wild (ANL)

Current problems of storage I/O software stack



- Optimization: complex stack, deep distributed storage hierarchy
- Coordination: poor state of programmable control mechanisms are not available (e.g., for data staging, dynamic load balancing, resilience)
- Exploit data locality

Outline

- ***Optimization*: Model-based autotuning of collective I/O**

- *Coordination*: Data staging coordination

- *Exploit data locality*: Improving the scalability and performance of the Swift workflow language by leveraging data locality through Hercules

Parallel I/O tuning

- Huge parameter space of the storage I/O software stack
- Domain knowledge is increasingly harder: software and hardware complexity

Application

High-Level I/O Library — Block alignment, chunking, etc.

I/O Middleware — Collective/independent I/O, data sieving, collective buffer size, number of aggregators, etc.

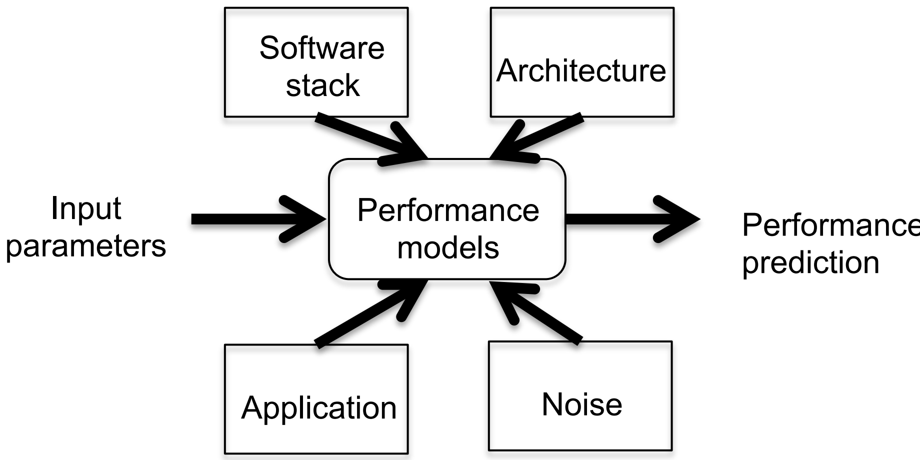
I/O Forwarding

Parallel File Systems — Striping unit, striping factor, striping layout, caching, prefetching, etc.

Storage drivers

Parallel I/O autotuning approaches

- Model based tuning
  - Analytical: Extensive domain knowledge required: software stack, architectural characteristics
  - Machine learning [Kumar2013, Yu2012]
- Search-based tuning
  - Genetic algorithms [Bezhad2013]
  - Simulated annealing [Chen2000]
- Hybrid [Bezhad2014]

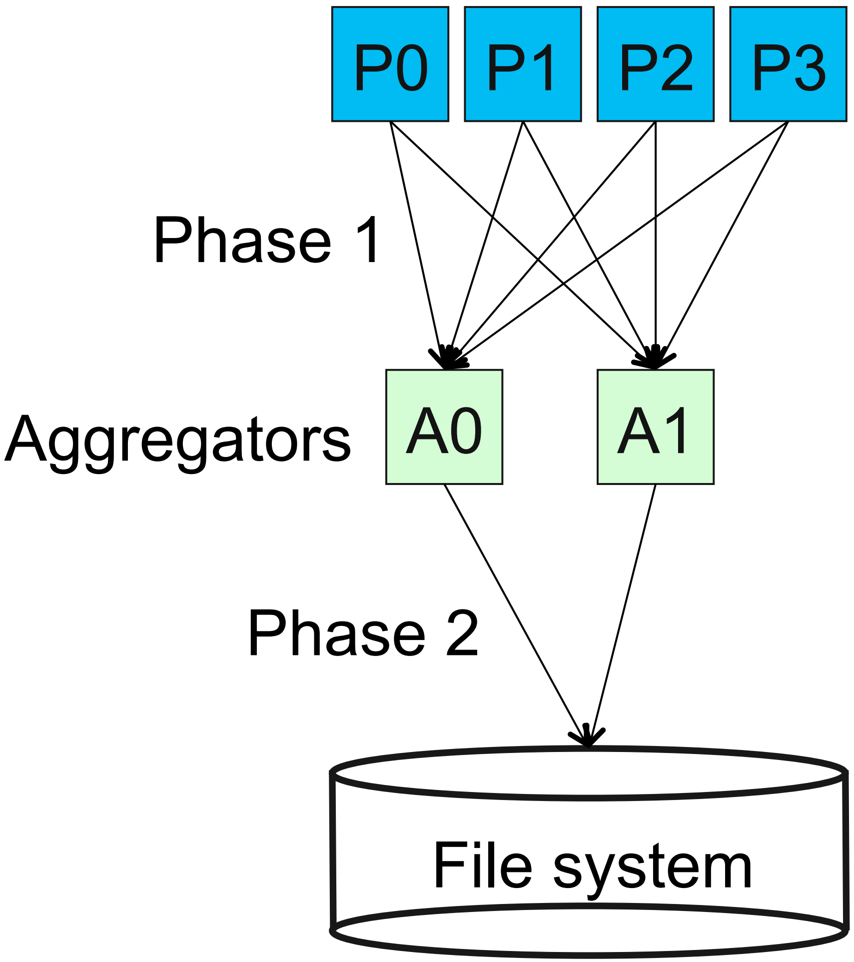


This work

- Model-based tuning of two-phase-I/O, the most popular collective I/O implementation from ROMIO
- Combination of analytical and machine learning models
- IEEE Cluster 2015 paper

Collective I/O

- N processes collectively write or read to a file
- Two-phase I/O write
  - Computation and communication for mapping writes to the file domain
  - Communication for sending data to aggregators
  - Storage I/O for storing the data to the file system



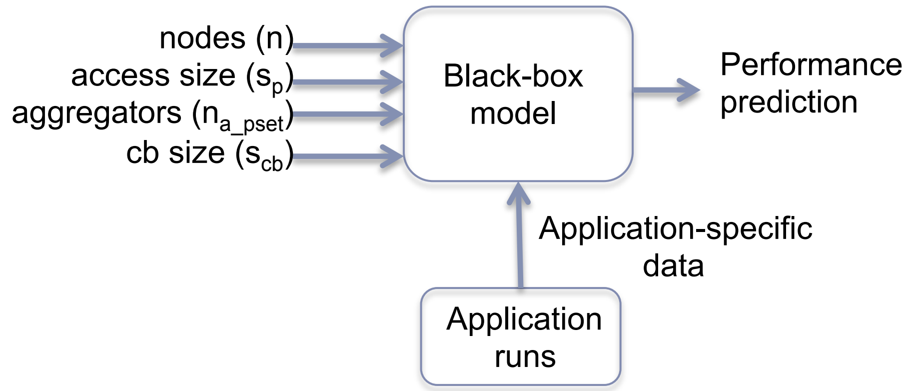
Modeling framework

4 parameters

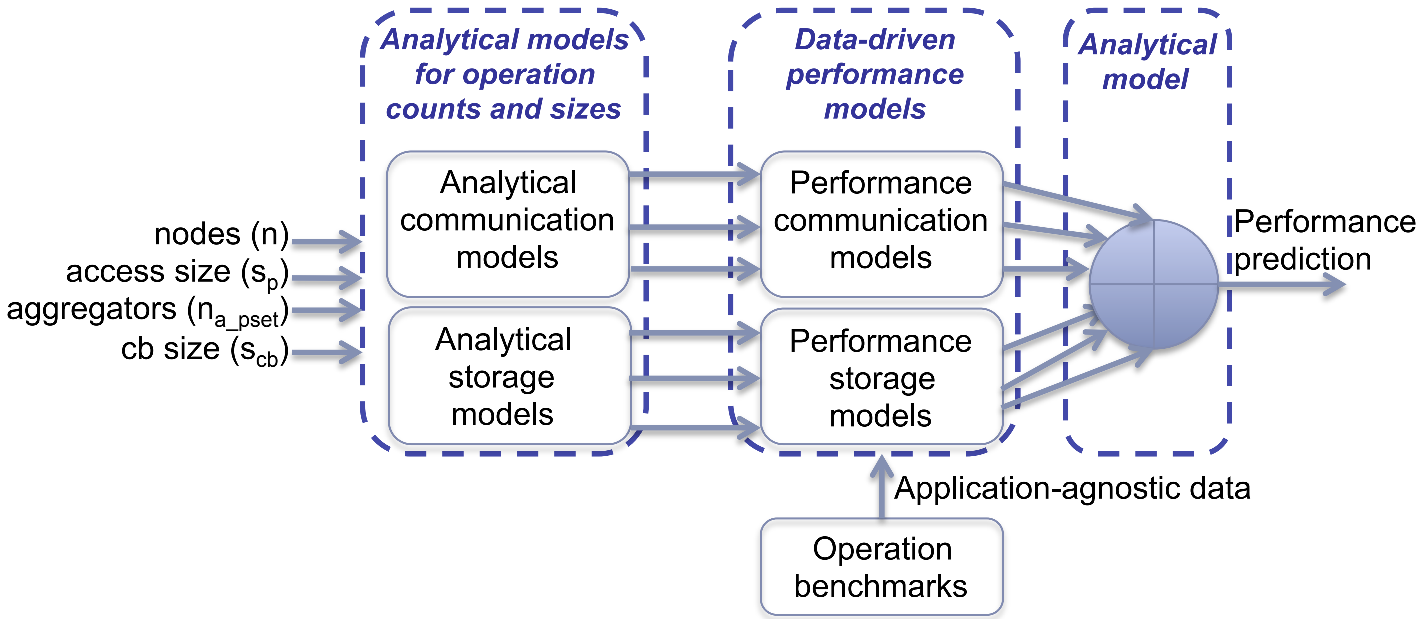
- n: number of nodes
- s: access size
- n_a: number of aggregators
- s_{cb}: collective buffer size

Goal: tune n_a and s_{cb}

Black box model



Hybrid model



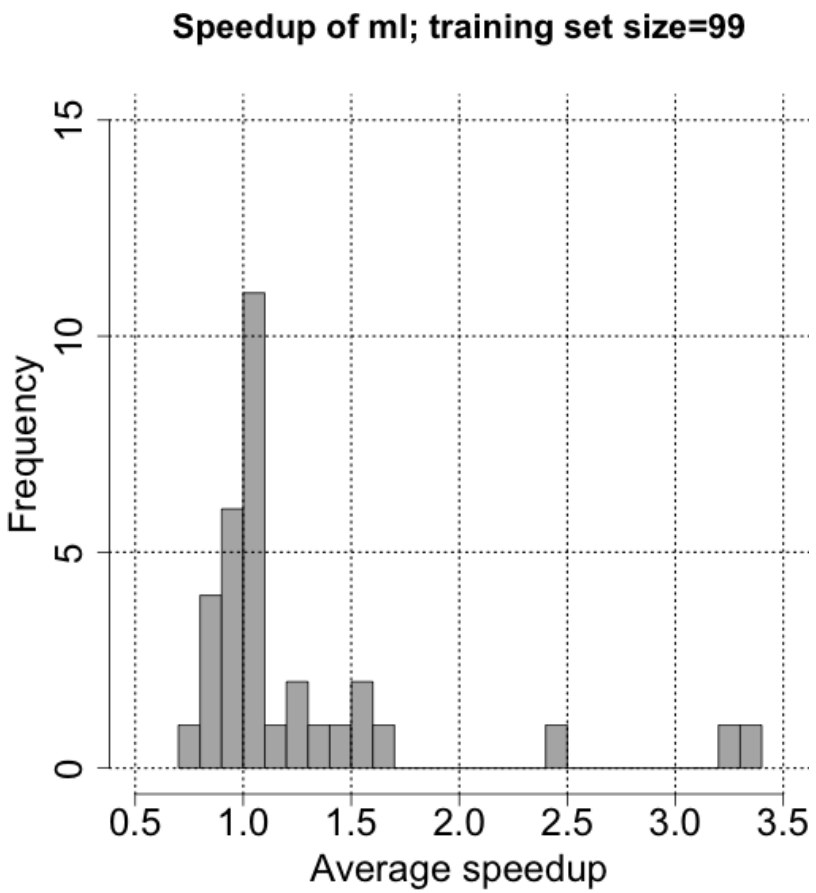
Experimental evaluation

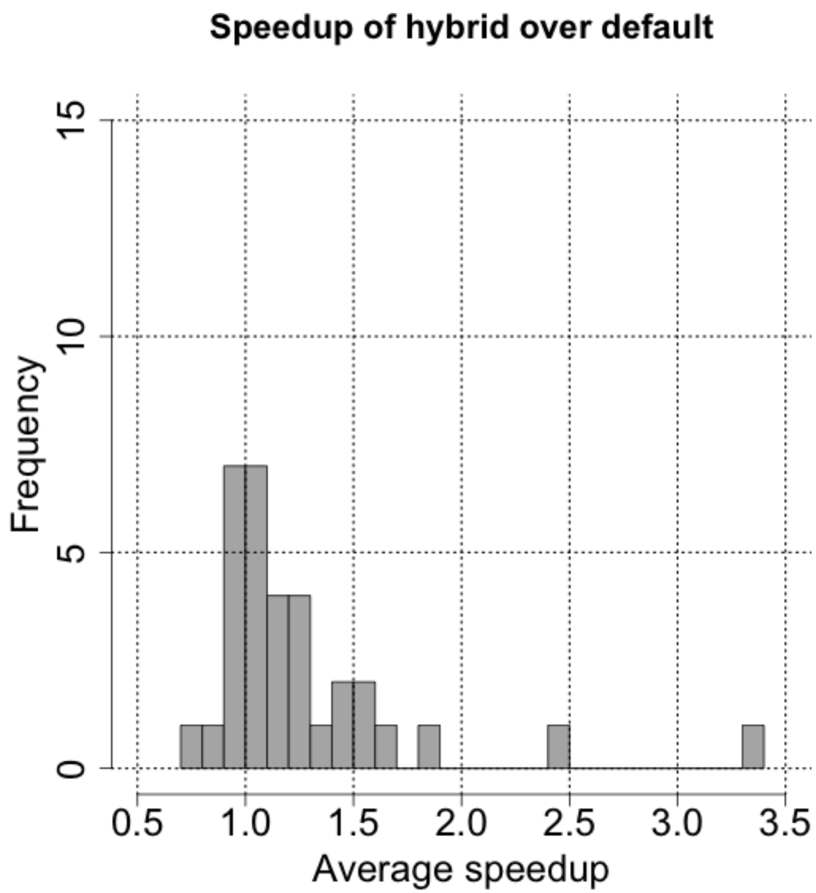
- IOR benchmark: N processes concurrently write and non-overlapping region to the file system through MPI-IO
- MPICH 3.1
- Vesta Blue Gene/Q
  - 2048 compute nodes 1.6 GHz 16 cores 16GB RAM
  - 5D torus network interconnecting compute nodes
  - 1 I/O node per 32 compute nodes
  - Client (I/O node) cache: 4GB
  - GPFS 3.5: Block size: 8MB, 40 NSD SATA data drives (Max throughput: 250 MB/s)
- Benchmark for performance models: ALCF MPI benchmark

Machine learning model

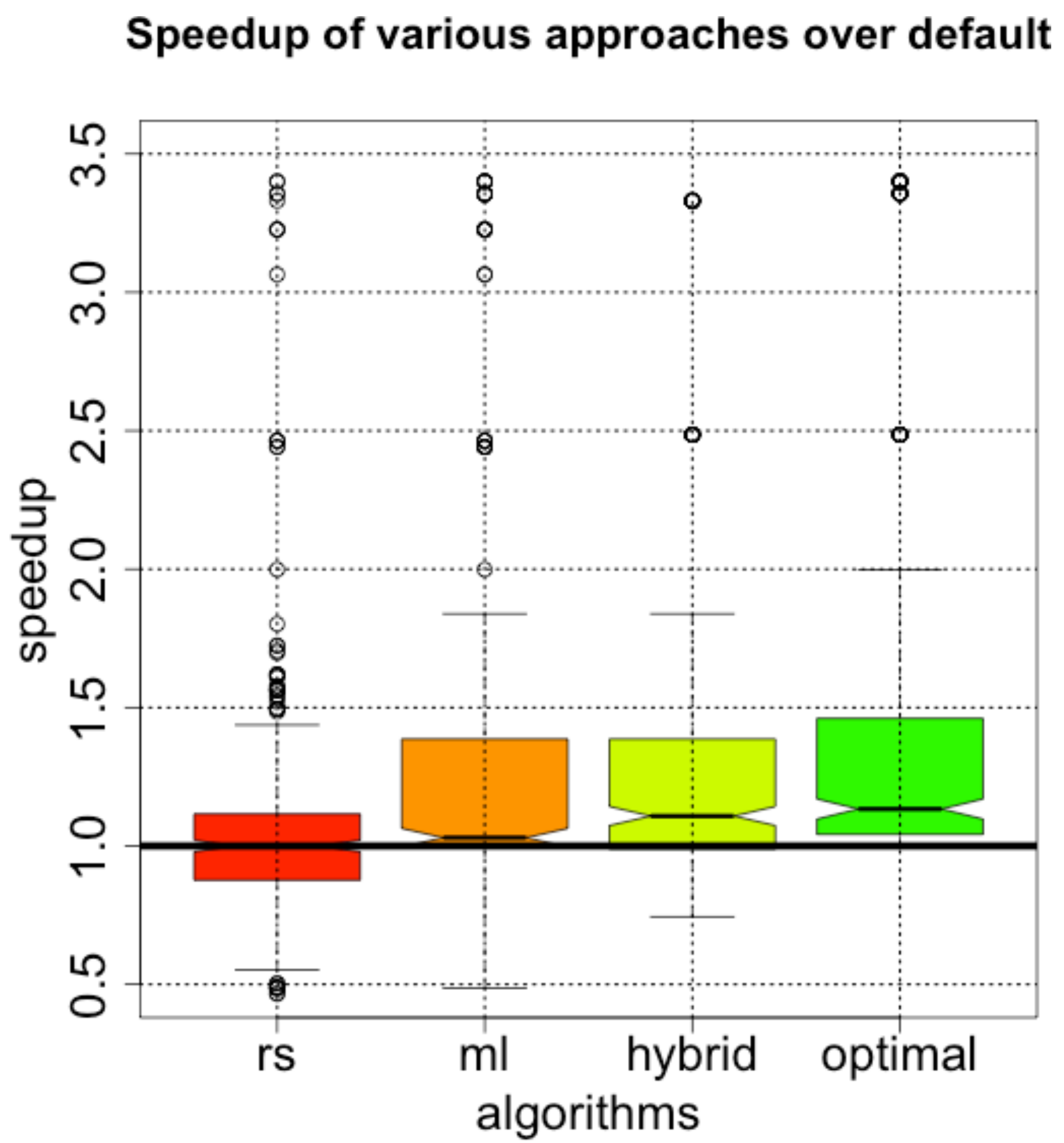
- Black box models and performance models
  - linear regression, neural networks, support vector machines, random forests, and cubist
  - Selected the model with best RMSE and R^2
- Data set:
  - Black box model: 297 points
    - Processes: 2048 (128 nodes on 16 cores), 4196, and 8392
    - Transfer sizes/core (MB): 1, 2, 4, 8,16, 32, 64, 96, 128, 192, 256
    - Collective buffer size (MB): 8, 16 (default), 32
    - Number of aggregators per 128 nodes: 40, 136, 520 (default)
  - Performance models
    - Alltoall: : 51 points for 2,048, 4,096, and 8,192 ranks and for message sizes between 1 byte and 256KB.
    - Alltoallv: 1,044 points for distributing message sizes between 1 byte and 64 MB (in powers of 2) for subsets of 2,048, 4,096, and 8,192 ranks.
    - Allreduce: 57 points for 2,048, 4,096, and 8,192 ranks and for message sizes between 4 bytes and 1 MB.
    - POSIX: 567 points for various sizes and various subsets of 2,048, 4,096, and 8,192 ranks.

Comparison between black-box and hybrid models

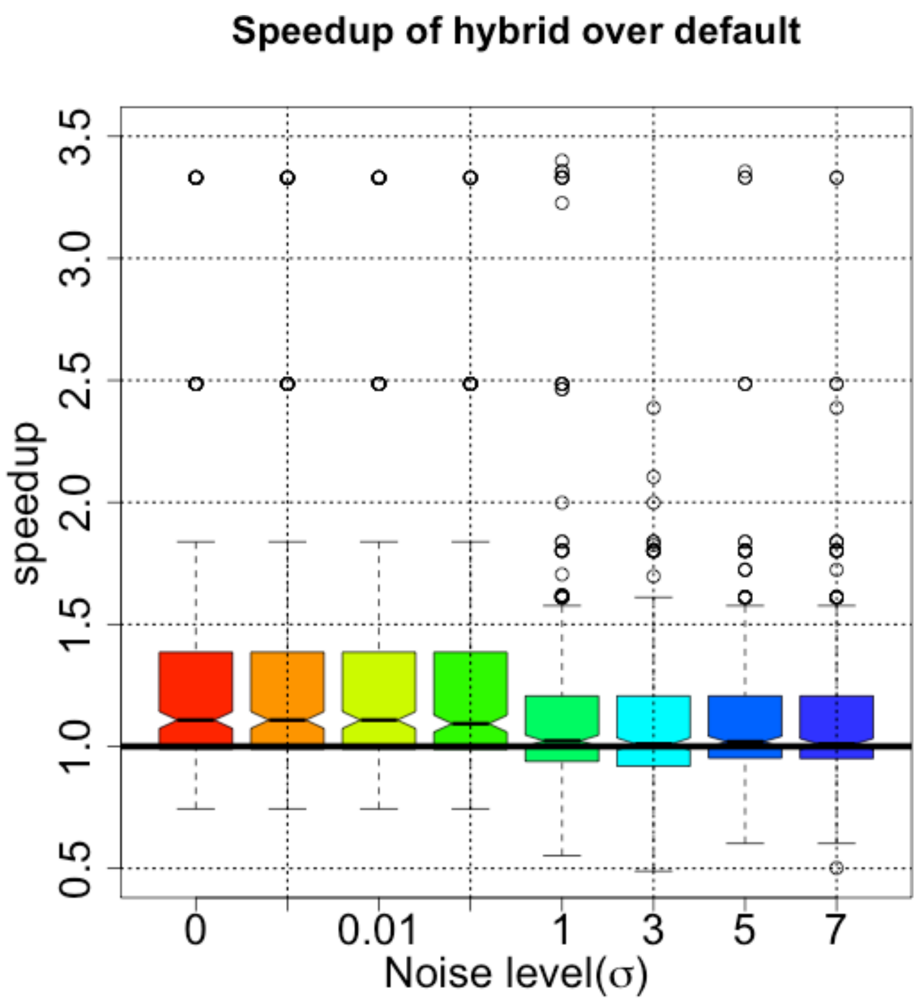


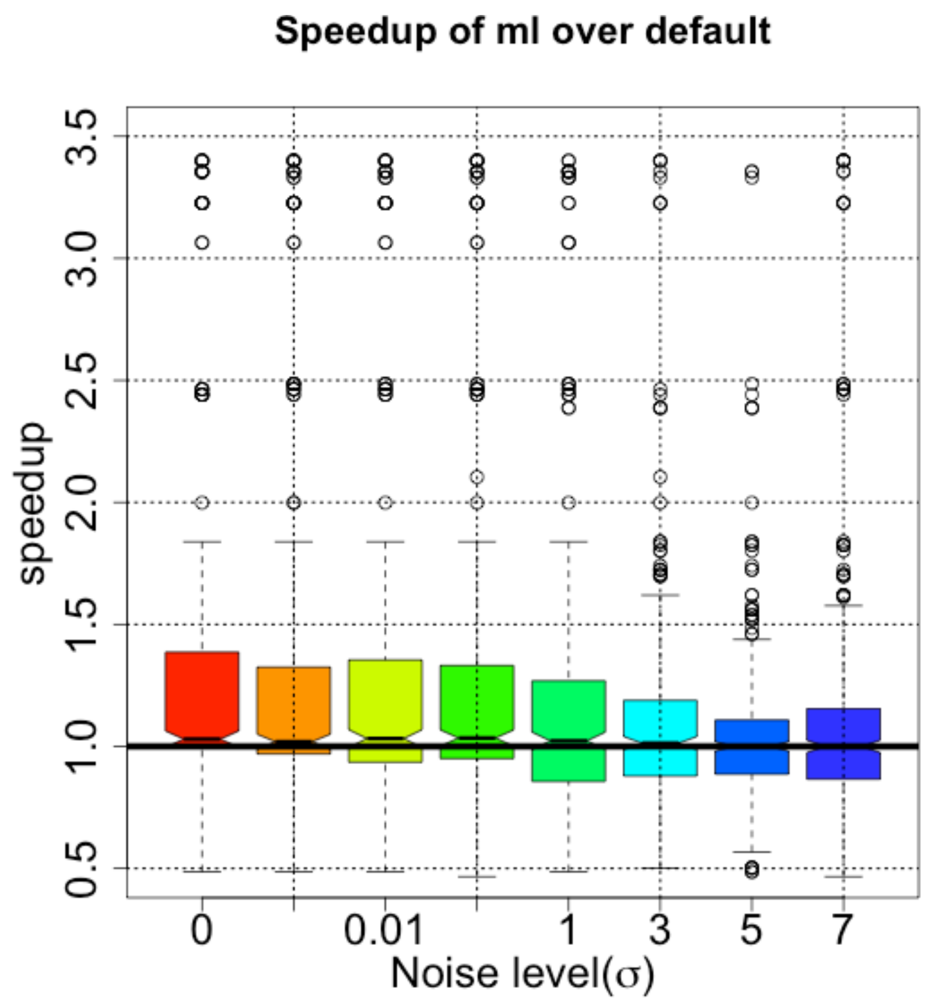


Comparison among various approaches



Impact of noise





Outline

- *Optimization*: Model-based autotuning of collective I/O

- ***Coordination*: Data staging coordination**

- *Exploit data locality*: Improving the scalability and performance of the Swift workflow language by leveraging data locality through Hercules

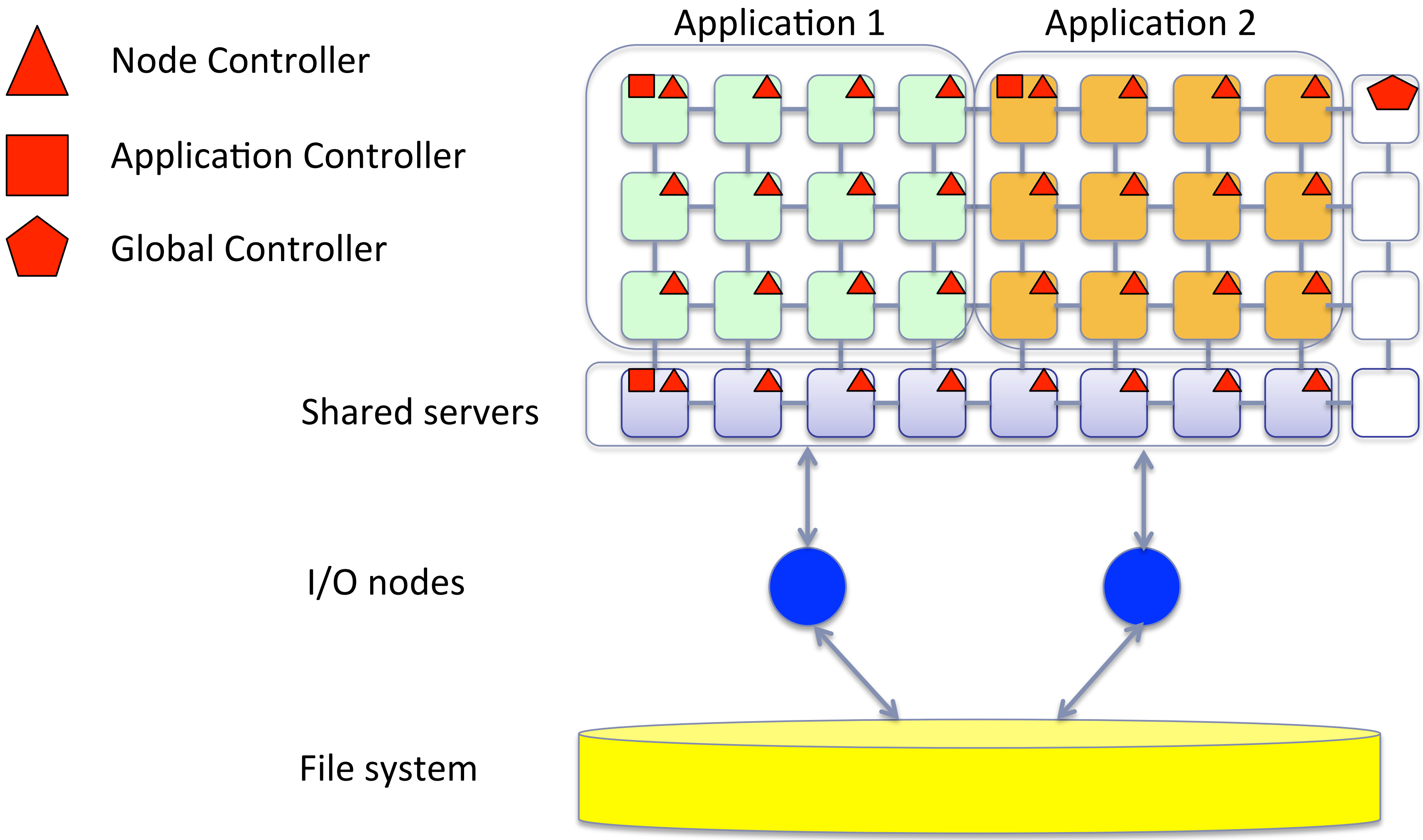
Data staging challenges

- Concurrent parallel data flows
  - Lack of data staging coordination
    - Among applications
    - Between applications and the system
- Increasing storage hierarchy
- Lack of standards for dynamic monitoring of large scale infrastructures (e.g. load, faults)
- Coupled control and data mechanisms
- Goal: offer novel mechanisms for data staging coordination to improve
  - Load balance
  - Resilience
  - Parallel I/O scheduling

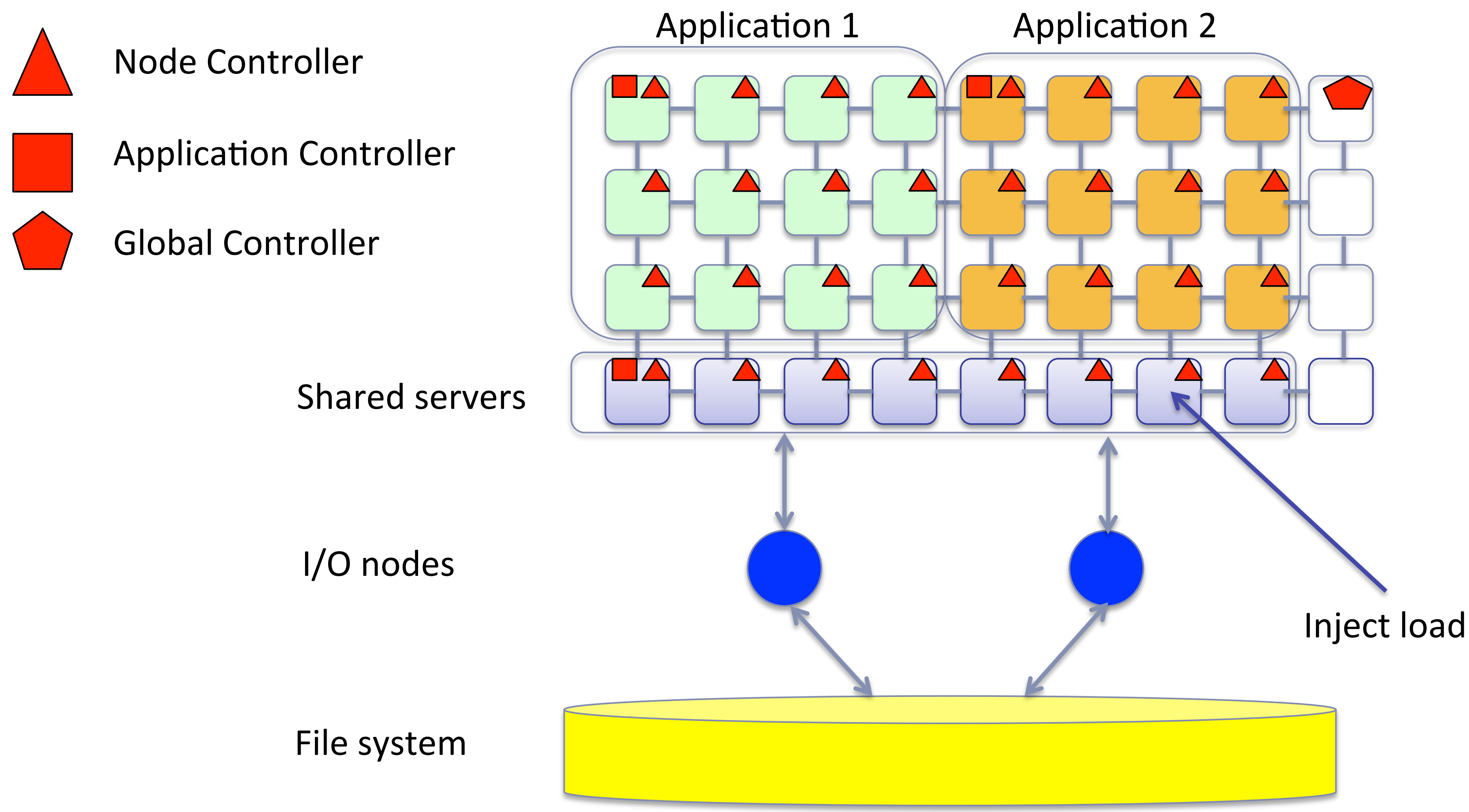
Coordination approach : CLARISSE library

- Decouple the data and control paths
- Data-path: abstractions used to implement data access operations
  - Collective I/O
    - 2 implementation: view based I/O, list-IO (can be used as both server-based I/O and client-based I/O)
- Control path: Based on a publish/subscribe substrate (e.g. Beacon)
  - Processes can subscribe to events having certain properties
    - Associate call-back
    - Wait for an event
    - Check for the arrival of an event
- Hierarchical control
  - Global controller
  - Application controller
  - Node controller
  - All nodes participate in control

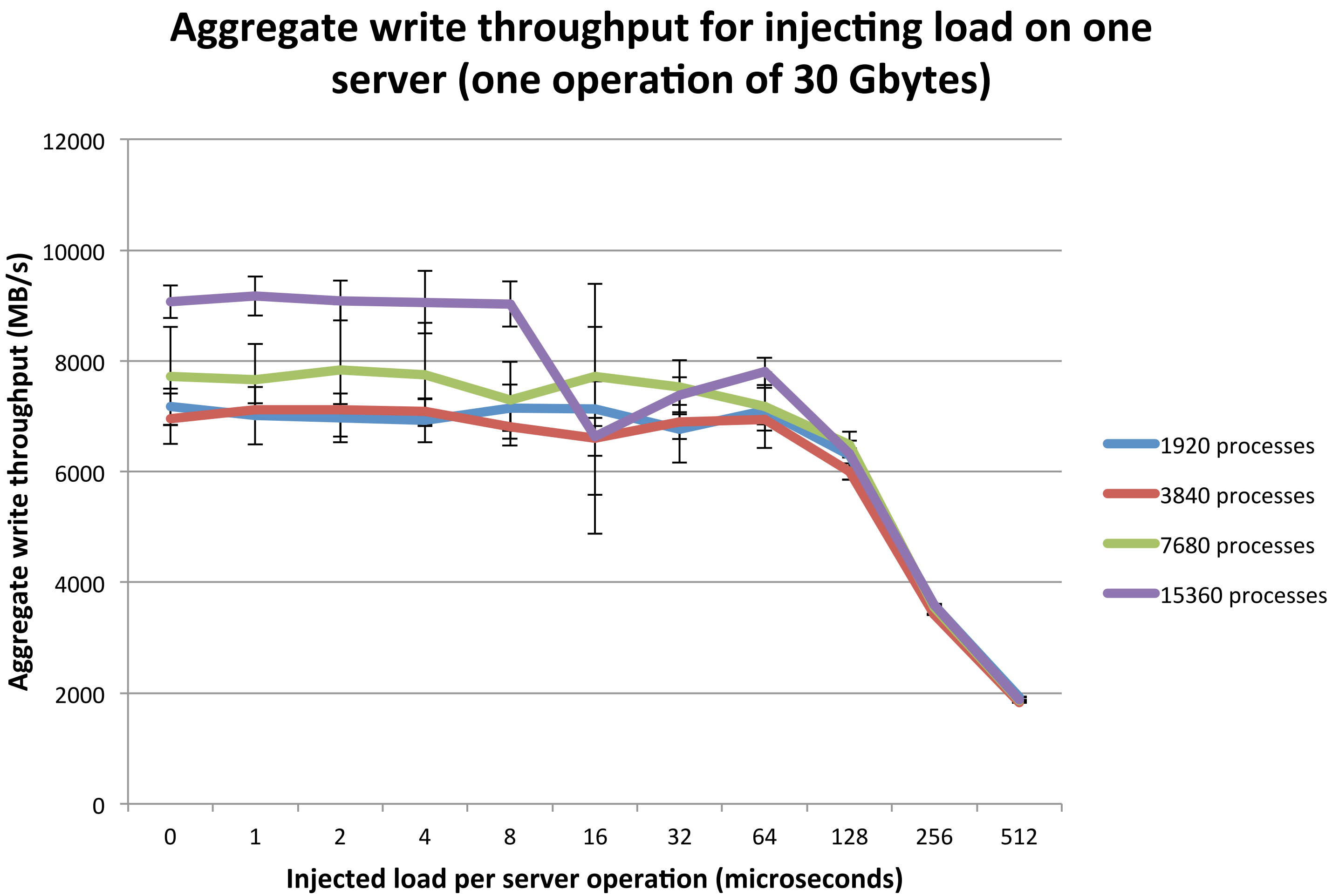
CLARISSE hierarchical control infrastructure



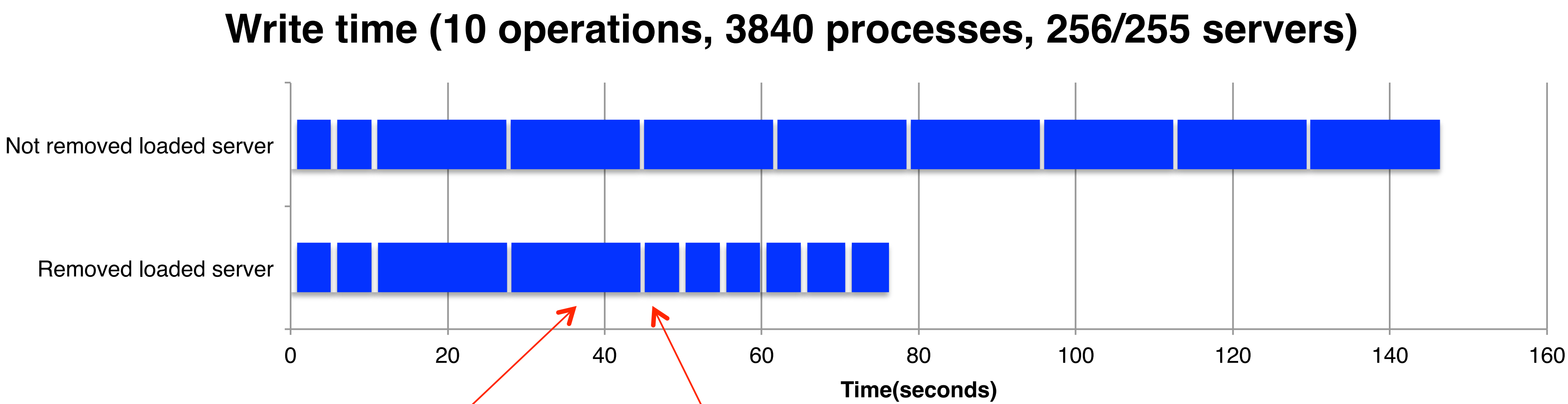
CLARISSE hierarchical control infrastructure



Load injection at 1 server (1 application)



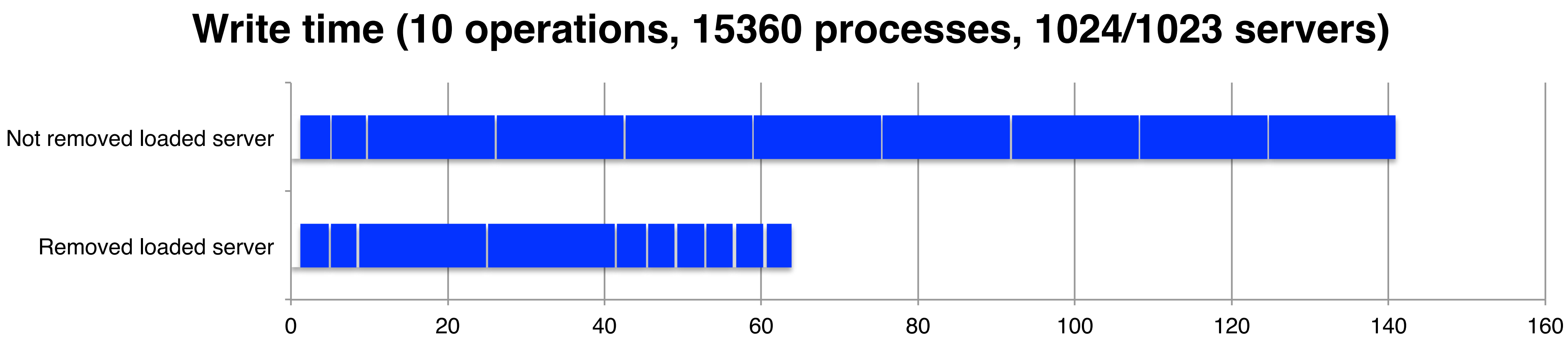
Dynamically scaling-down



Detect loaded server
Reconstruct server map

New epoch with fewer servers

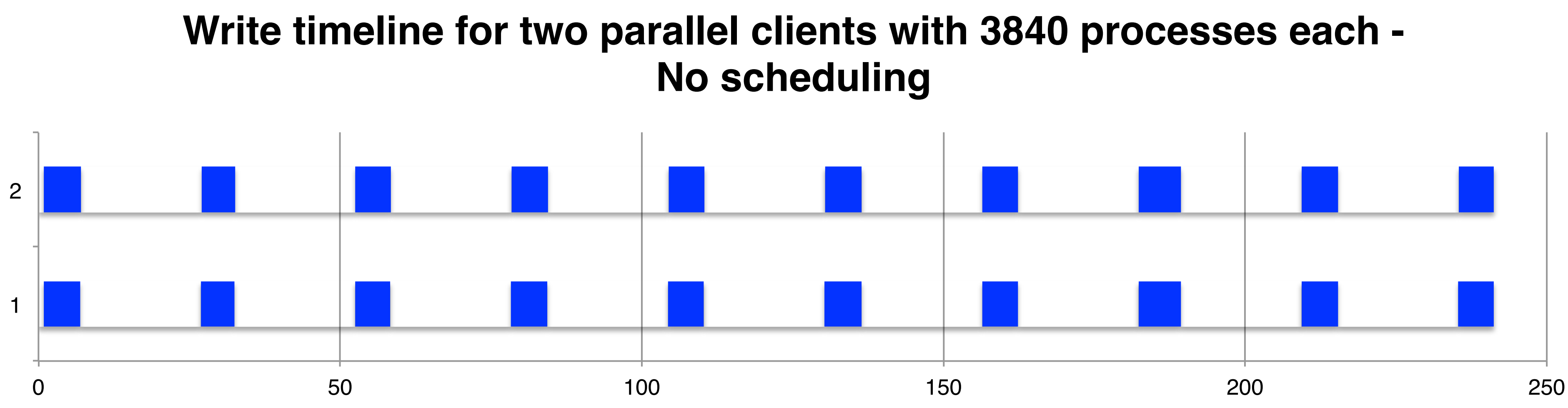
Dynamically scaling-down

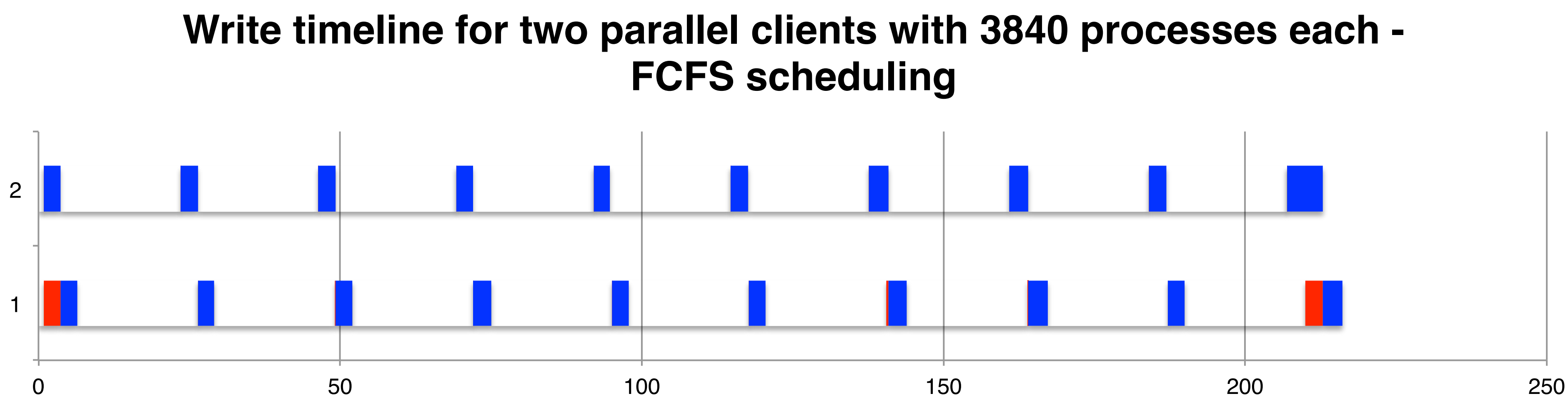


Parallel I/O scheduling

- Several applications share
- The application controller notifies the global controller
- The global controller schedules the next application to be run
- Several policies possible
  - FCFS evaluation

FCFS scheduling versus no scheduling





Outline

- *Optimization*: Model-based autotuning of collective I/O

- *Coordination*: Data staging coordination

- ***Exploit data locality*: Improving the scalability and performance of the Swift workflow language by leveraging data locality through Hercules**

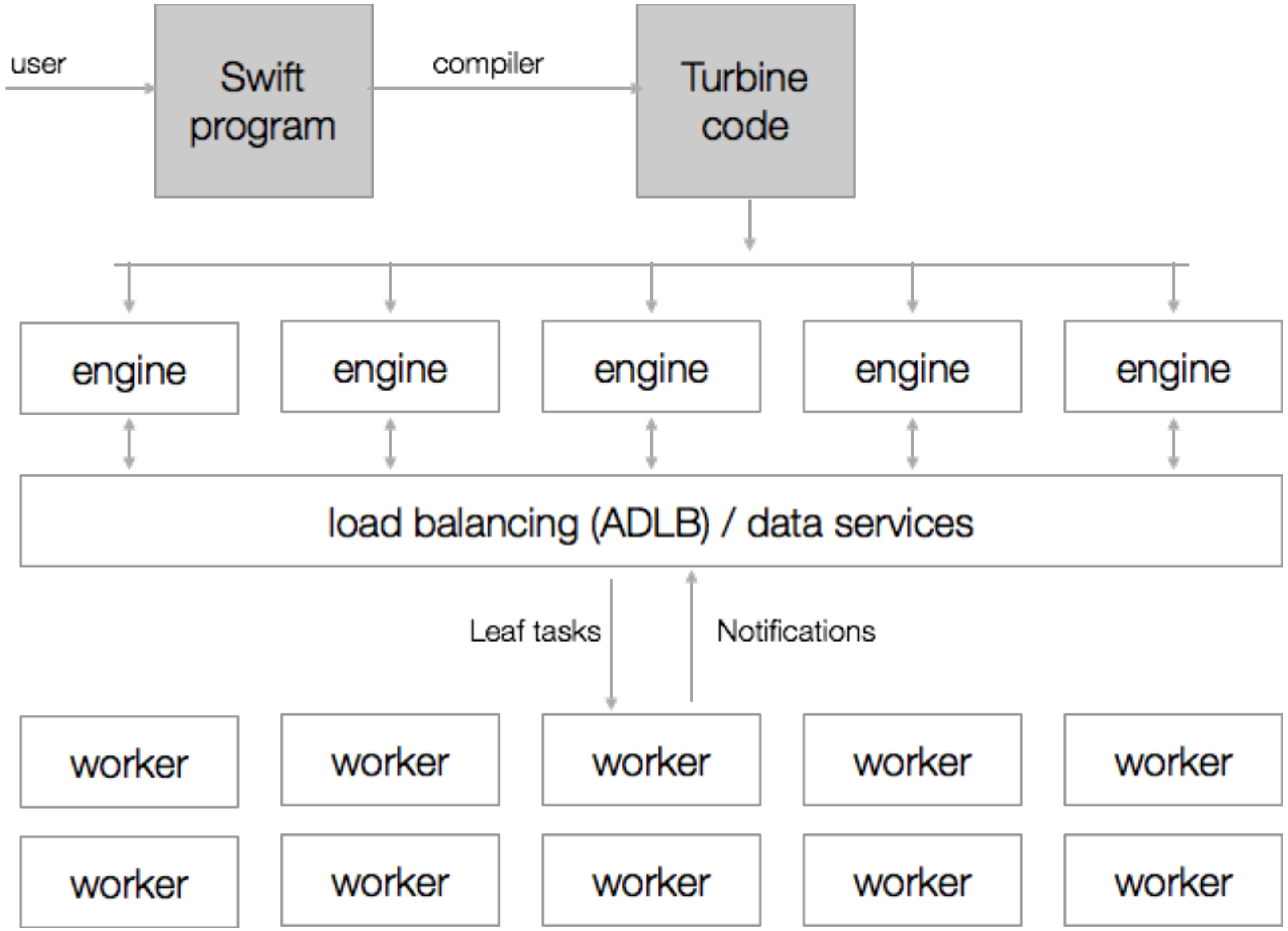
Swift/T

- Swift/T: Language and runtime for dataflow applications

```
(int r) myproc (int i, int j)
{
    int f = F(i);
    int g = G(j);
    r = f + g;
}
```

- F( ) and G( ) implemented in native code or external programs
- F( ) and G( ) run concurrently in different processes
- r is computed when they are both done

Swift/T architecture

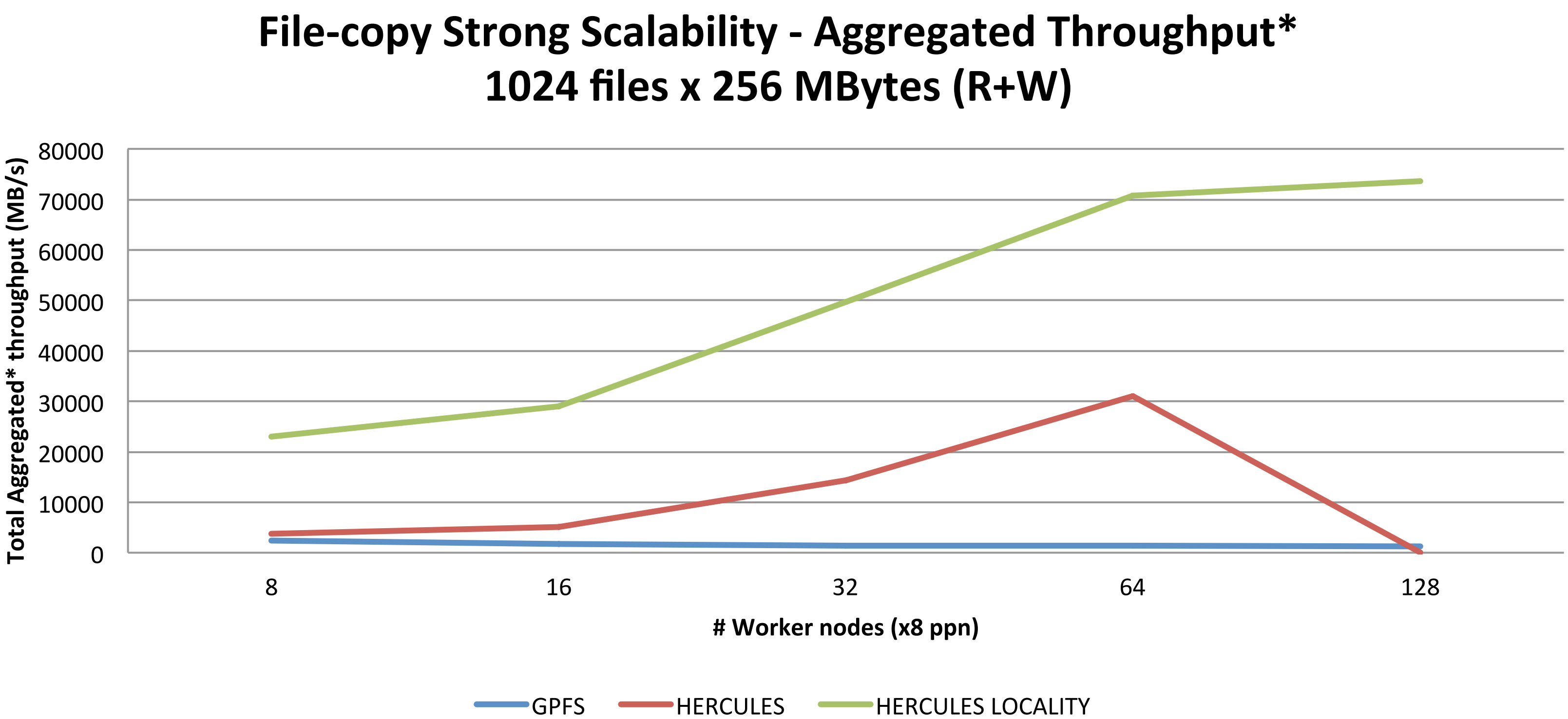


Problem description

- Load balancer is not locality-aware
- Tasks communicate through the parallel file system (bottleneck)
- Objectives:
  - Improve the performance of inter-task communication
    - Data locality
  - Investigate the tradeoffs between data locality and load-balance in workflow execution
    - Ideal load balance, but poor locality
    - Ideal data locality, but poor load balance (not all nodes used)

Approach

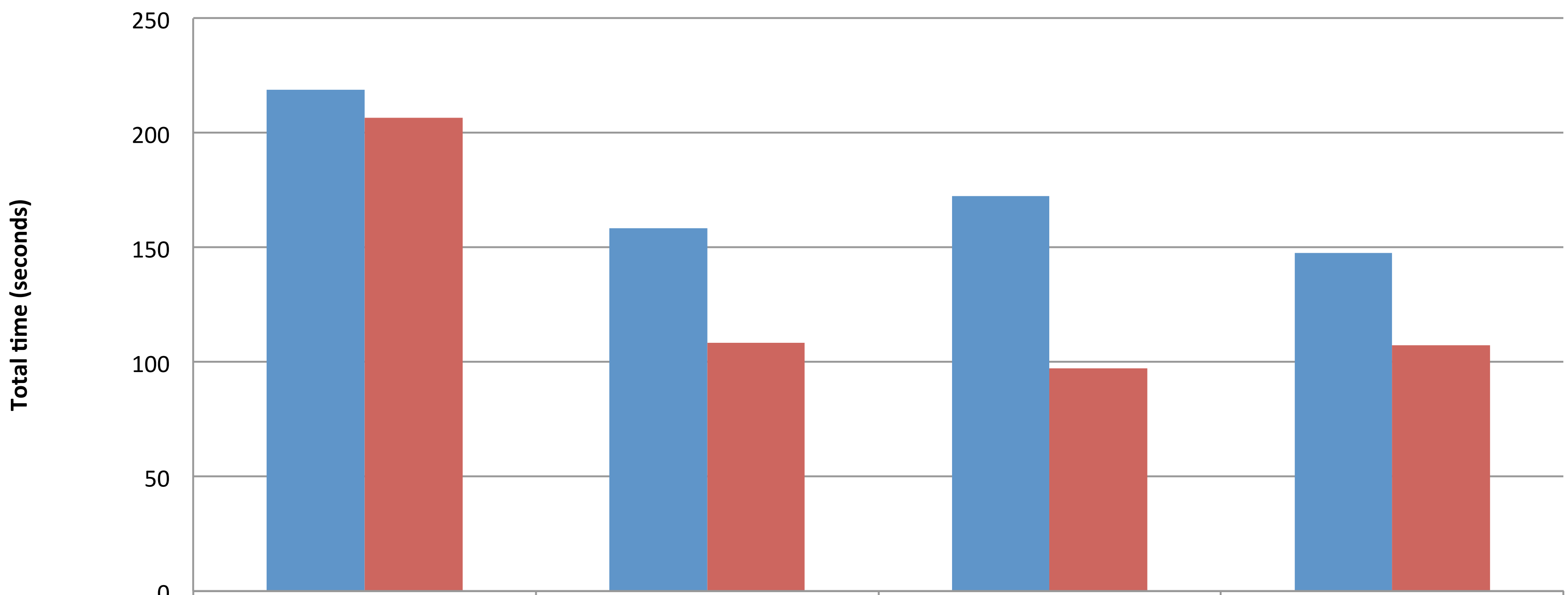
- Hercules
  - persistent key value store based on Memcached
  - On-demand deployment of servers on application nodes
- Data placement over the servers
  - Consistent hashing (original Memcached)
  - Locality-aware (implemented)
  - Load-aware (under implementation)
  - Capacity aware
- New Swift language constructs
  - Soft location: best effort task placement
  - Hard location: enforce data locality



Fusion Linux cluster

- 320 nodes, 2 x quad core, 36 GB RAM
- Infiniband QDR (4 GB/s) and gigabit ethernet
- GPFS: up to 2500 MB/s

MapReduce-like WC application
256 files x 256 MB - 64 GB
Total execution time



	GPFS	Hercules	Hercules Locality - SOFT	Hercules Locality HARD
8x8 workers	218.731	158.384	172.57	147.741
32x8 workers	206.769	108.34	97.32	107.45

Ongoing and future work

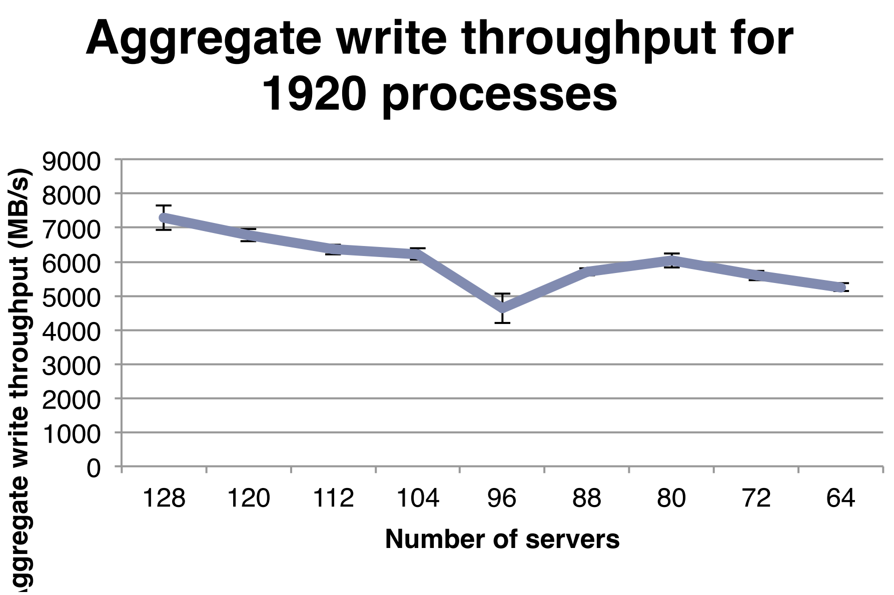
- Model-based autotuning I/O
  - Performance predictability
  - Improve individual models
  - Noise
    - Load and noise modeling for load detection in data staging (multiple servers)
- Data staging coordination
  - Topology-aware server/aggregator placement - JL Colaboration with E. Jeannot, F. Tessier (INRIA), V. Vishnavath (ANL)
  - Multiple stage coordination (aggregation – burst buffer – file system)
  - Load prediction based on Omnisc'IO (Mathieu Dorrier – ANL)
  - Adaptive buffering in parallel applications workflows (Decaf project)
  - Adopt Global Information Bus from Argo and Hobbes (Beacon, Exposé)
  - Need for sub-second monitoring and notification
- Exploit locality in workflows
  - Load-aware placement
  - Tradeoff locality – load balance
- New applications? New architectures? New coordination scenarios?

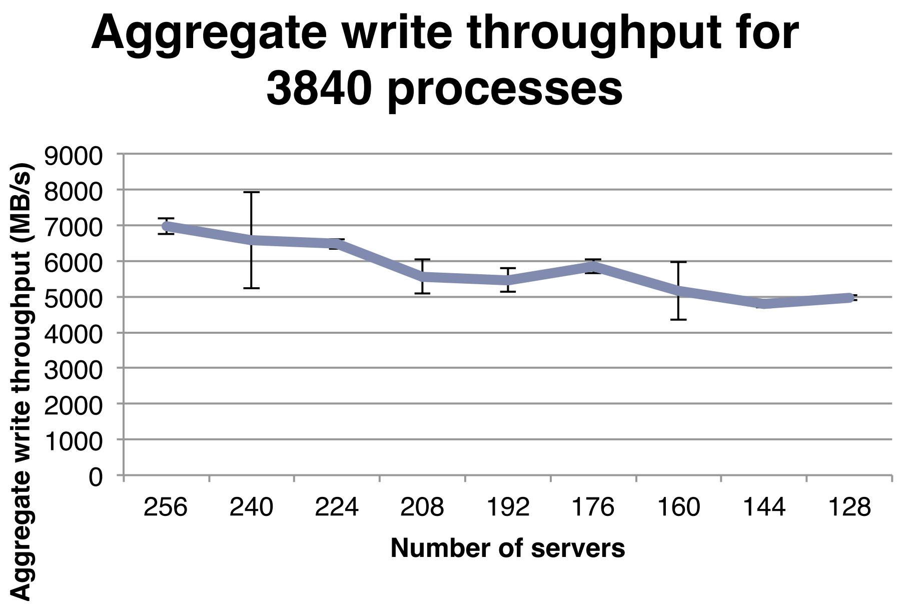
Thank you

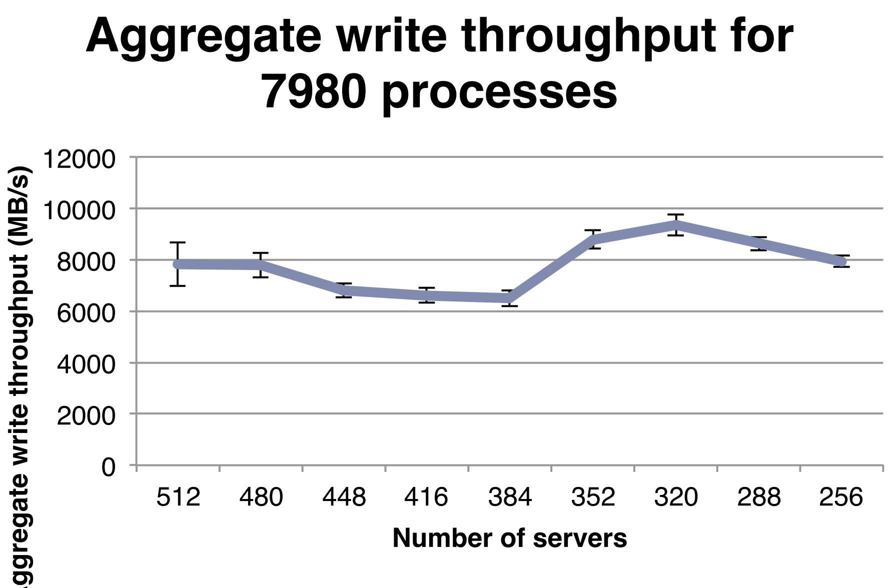
Conclusions - autotuning

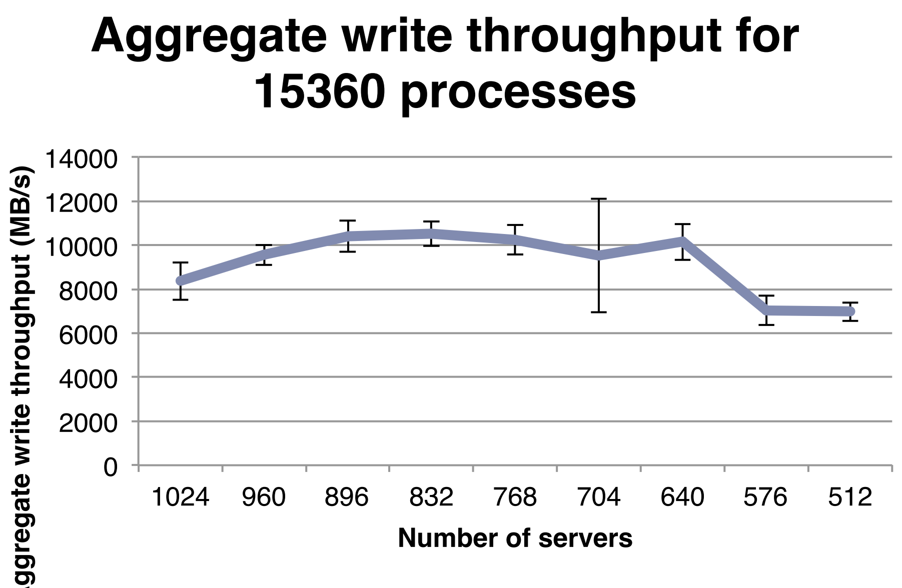
- Automatic parameter configuration
  - Machine learning and hybrid models approaches outperform the default values in most cases
  - Hybrid models higher robustness to noise than pure machine learning
  - Hybrid model do not require application reruns
- Factors that limit efficiency of the I/O stack optimization
  - POSIX consistency semantics: File locking
  - File system noise
  - The lack of information about the state of storage hierarchy (e.g. cached versus non-cached)
  - Performance predictability needs to improve

Scale-down number of servers (1 application)









Dynamic removal of loaded server

- Assumes the availability of a load detection mechanism
- One application process detects a loaded server
- Notifies the application controller
- Application controller informs all node controllers and ask them to prepare to start a new epoch with less servers
- Node controller
  - Decides the last operations to be executed from the current epoch
  - Suspends all operation from the future epoch
  - Updates the server map
  - Notifies the application controller
  - Application controller ask all nodes to start a new epoch
- Each node controller resumes the suspended operations if any

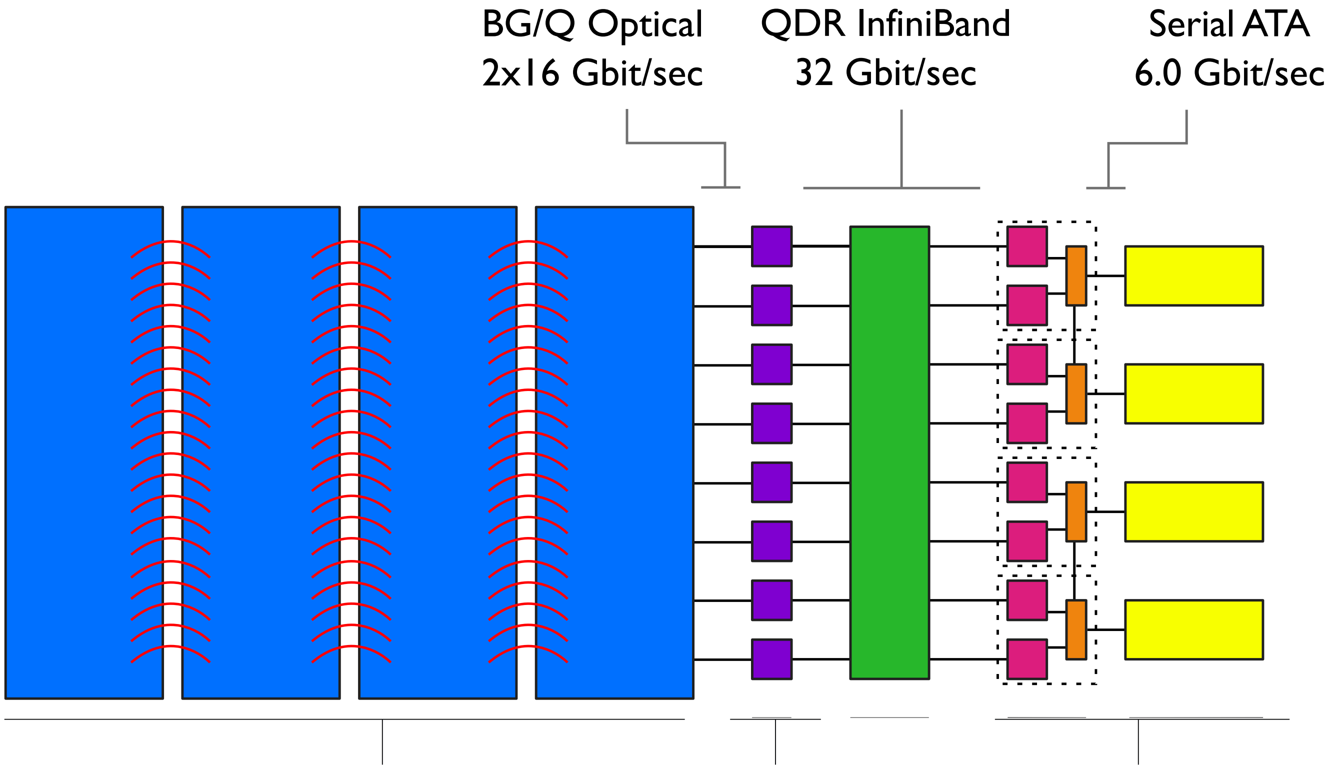
Conclusions - coordination

- Data staging coordination
- Separation of data and control
- Hierarchical controlling
- Significant benefits
  - Load/Fault aware sever-scale down
  - Parallel I/O scheduling
- Scalable load and fault monitoring is required

Conclusions – data locality

- Integration Swift/T - Hercules
- Substantially improves the throughput over shared file systems
- I/O performance scales up with the number of application nodes
- Exploit data locality in workflows
- Less sensitive to file system noise and contention

Vesta Blue Gene/Q system at ANL



2048 compute nodes 1.6 GHz 16 cores 16GB RAM

5D torus network interconnecting compute nodes

1 I/O node per 32 compute nodes

Client (I/O node) cache: 4GB

GPFS 3.5

- Data: 40 NSD SATA drives
- Max disk throughput: 250 MB/s (cache 1000 MB/s)
- Block size: 8MB