CLARISSE: Cross-Layer Abstractions and Run-time for I/O Software Stack of Extreme-Scale Systems

Florin Isaila

ANL & University Carlos III

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

Argonne NATIONAL LABORATORY

ARCOS

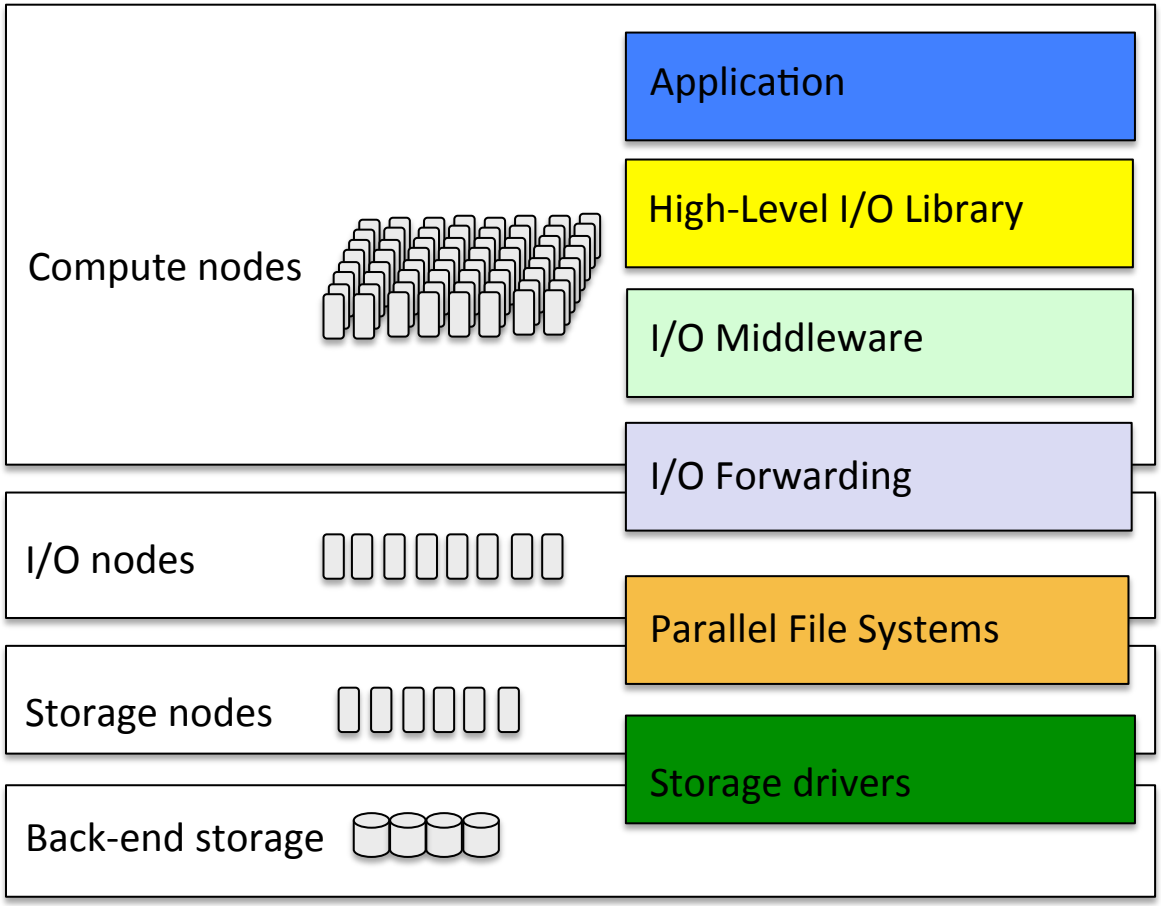
Outline

- Introduction
- CLARISSE overview
- Model-based I/O stack optimization
- Conclusions

Introduction

- Scientific applications (climate, genomics, high energy physics, etc.) process increasingly larger data sets
- Future high scale supercomputers need to deal efficiently with big data
- I/O software stack needs to evolve in terms of performance, programmability, resilience, energy efficiency
- This talk will concentrate on performance

Current problems of storage I/O stack



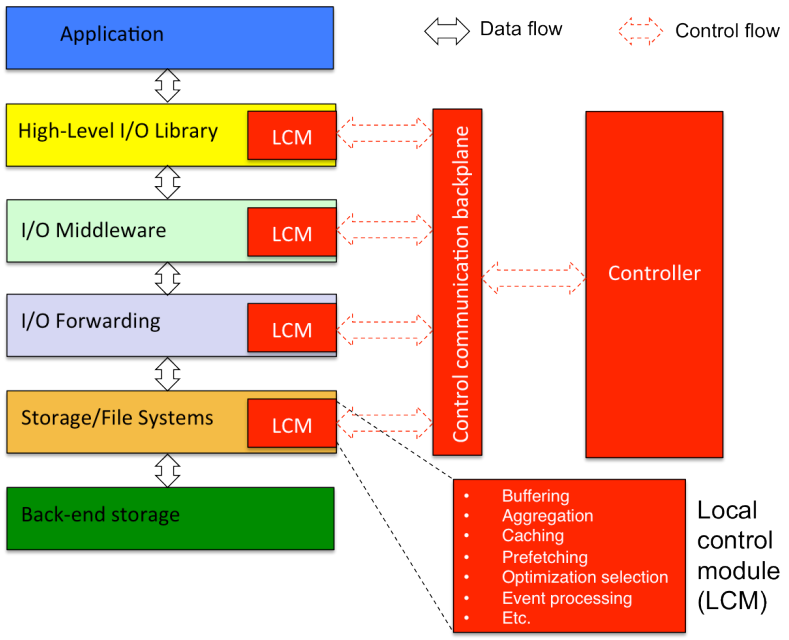
- Long path from compute nodes to final storage impacts performance (latency, throughput)
- Storage I/O optimizations are local: Difficult to perform global optimizations
- Cross-layer control mechanisms are not available (e.g., for data staging, dynamic load balancing, resilience)

CLARISSE overview

- Cross-Layer Abstractions and Runtime for I/O Software Stack (CLARISSE)
  - A 3-year project started October 2013
  - European “Marie Curie” International Outgoing Fellowship
  - Collaboration between ANL and UC3M (Spain)
- Goals
  - Enable global optimizations of the software I/O stack
  - Design novel cross layer control abstractions and mechanisms for supporting data flow optimizations
    - Collective I/O, data staging, exploit data locality

CLARISSE overview

- Global logical view of optimizations space
  - Distributed application of global optimization
  - Facilitate the combination of local optimizations based on a global view
- Cross-layer abstractions and run-time
  - Facilitate the flow of control and data across the I/O stack
  - Decouple the data and control planes
    - Control backplane (e.g. Argo project)
    - Data plane (e.g. Mercury)
- Data flow optimizations
  - Coordinated cross-layer buffering, caching, prefetching
  - Run-time optimizations (e.g. load-aware data staging)



Related work

- Software Defined Networking (e.g. Open Flow): global control based on separation of control and data flow
- I/O Flow (Microsoft Research): A Software Defined Storage Architecture for virtualized data centers
- Fast Forward (Intel et al.): redesign of the storage I/O stack
- Argo (ANL et al.), Hobbes (Sandia et al.): system software for exascale based on an OS/Run-time environment
- Cross-layer optimizations for current I/O stack
  - Parallel I/O autotuning (UIUC & LBNL)
  - Reduce performance interference (CaLCioM, INRIA & Argonne):

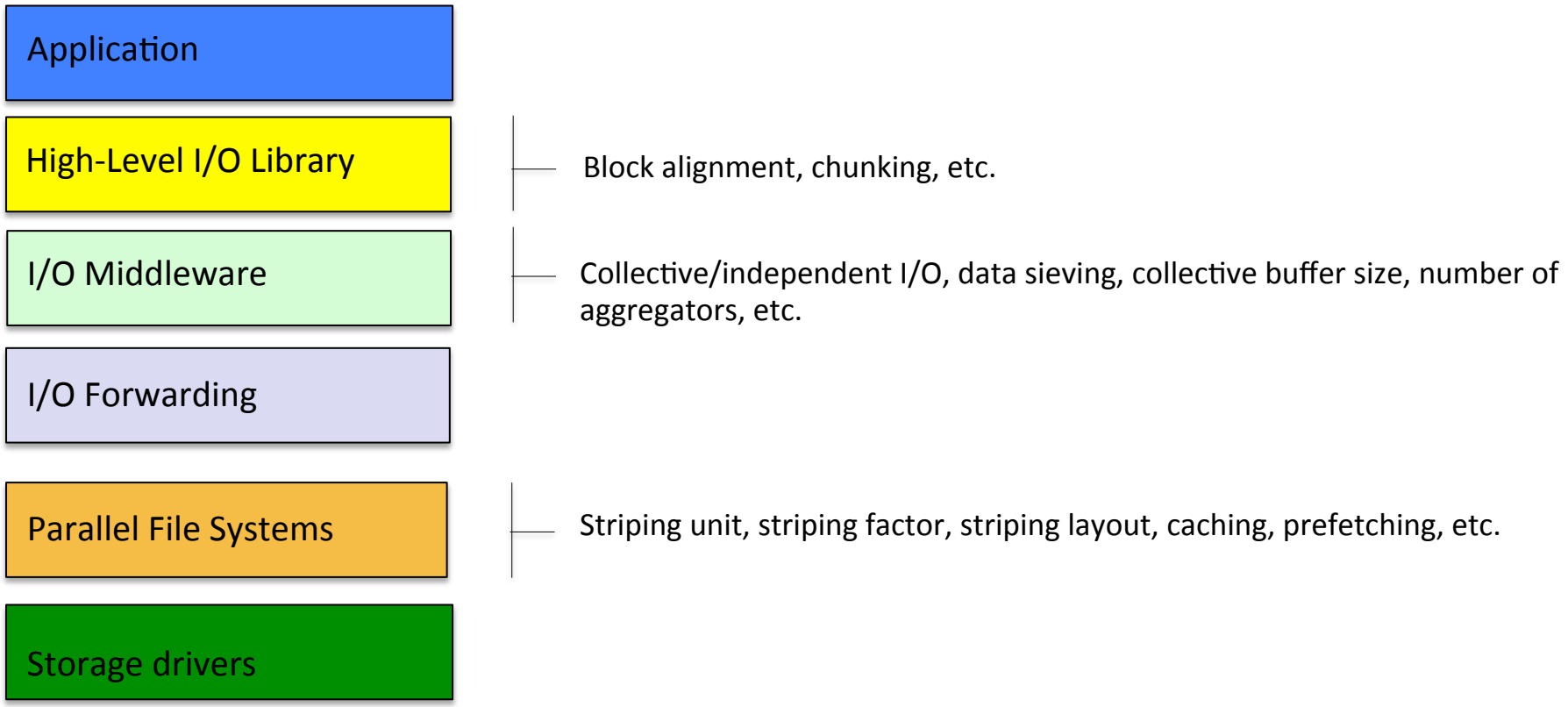
Outline

- Introduction
- CLARISSE overview
- **I/O stack optimization**
- Conclusions

Optimizing the I/O stack

- How complex is to optimize the current I/O stack?
- How does storage I/O use system resources?
- How predictable is the performance?
- Where are the inefficiencies?
- Which are the causes?
- What do we need to address?

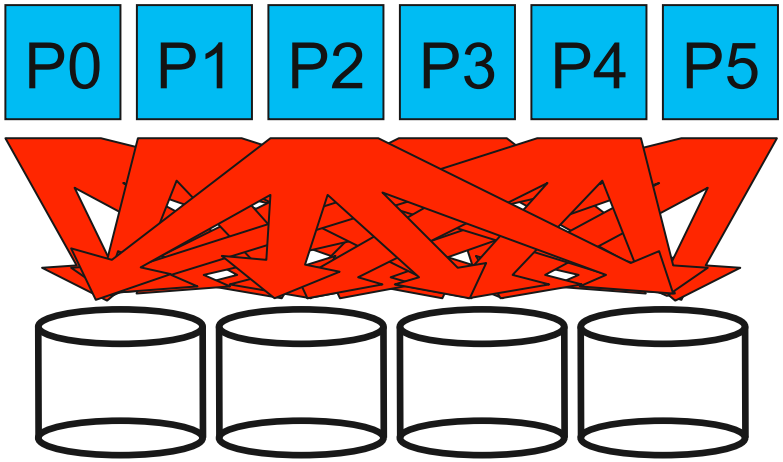
Current I/O stack optimization



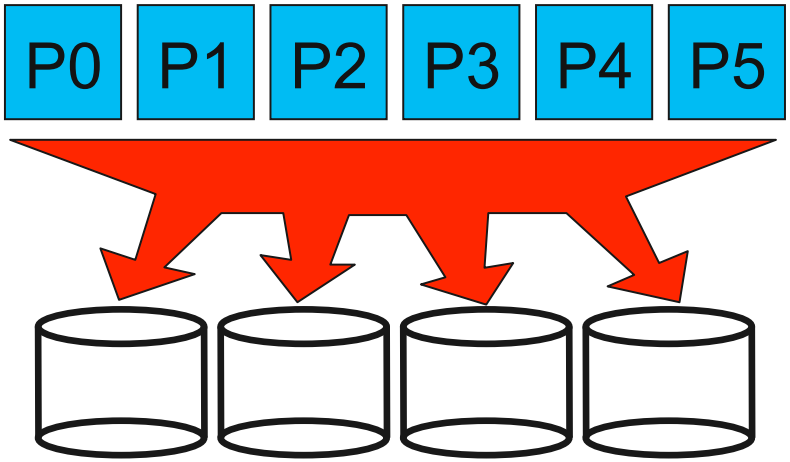
- Huge parameter space
- How can the optimization be approached?
  - domain knowledge, black-box, combination of the two
- Domain knowledge is increasingly harder
  - Which are the hurdles?

Motivational experiment

- IOR benchmark: N processes concurrently write and non-overlapping region to the file system through MPI-IO (MPICH 3.1)
- MPICH 3.1
- Darshan 2.2.8 HPC I/O characterization tool
- Access methods
  - Independent I/O shared file
  - Independent I/O file per processs
  - Collective I/O

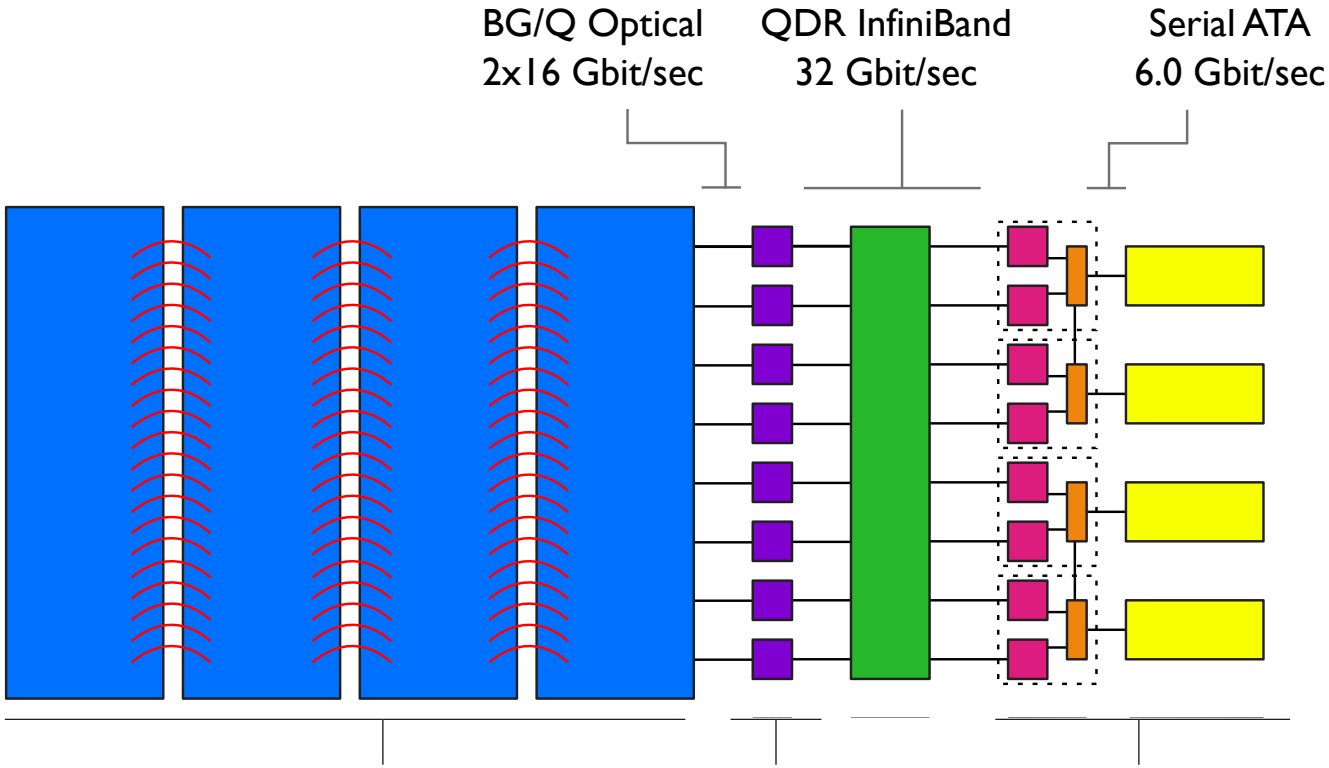


Independent I/O



Collective I/O

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

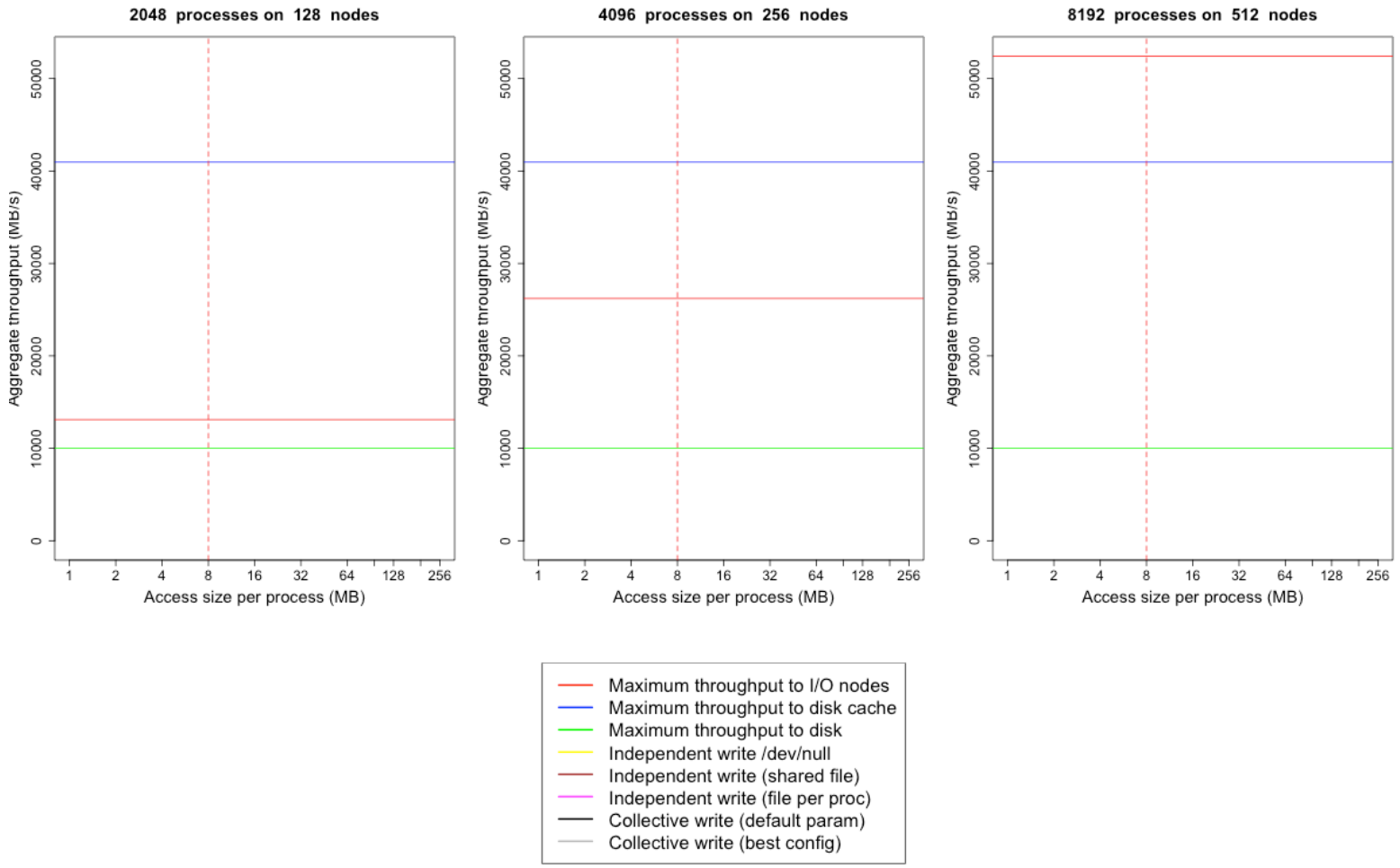
GPFS 3.5

Data: 40 NSD SATA drives

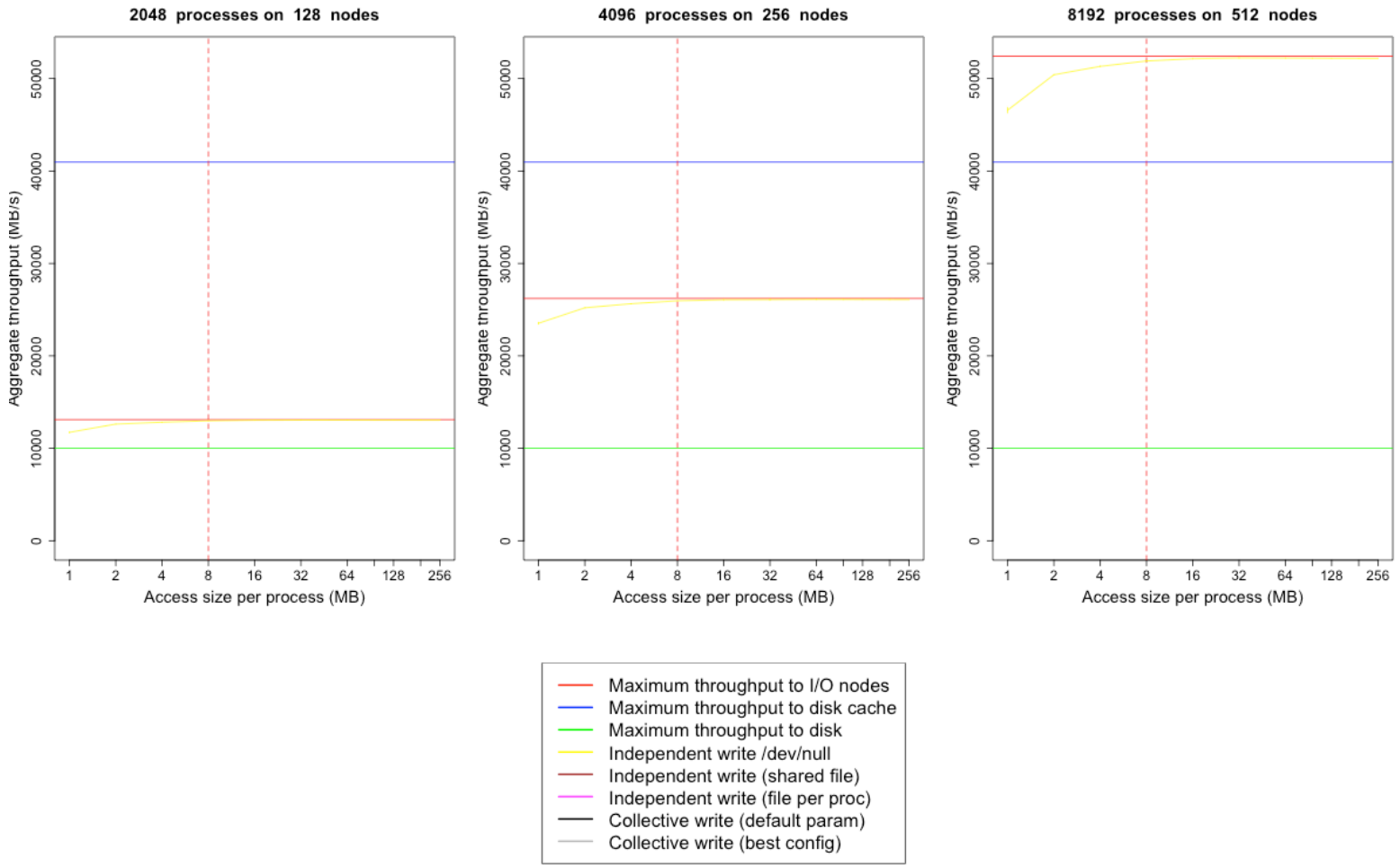
Max disk throughput: 250 MB/s (cache 1000 MB/s)

Block size: 8MB

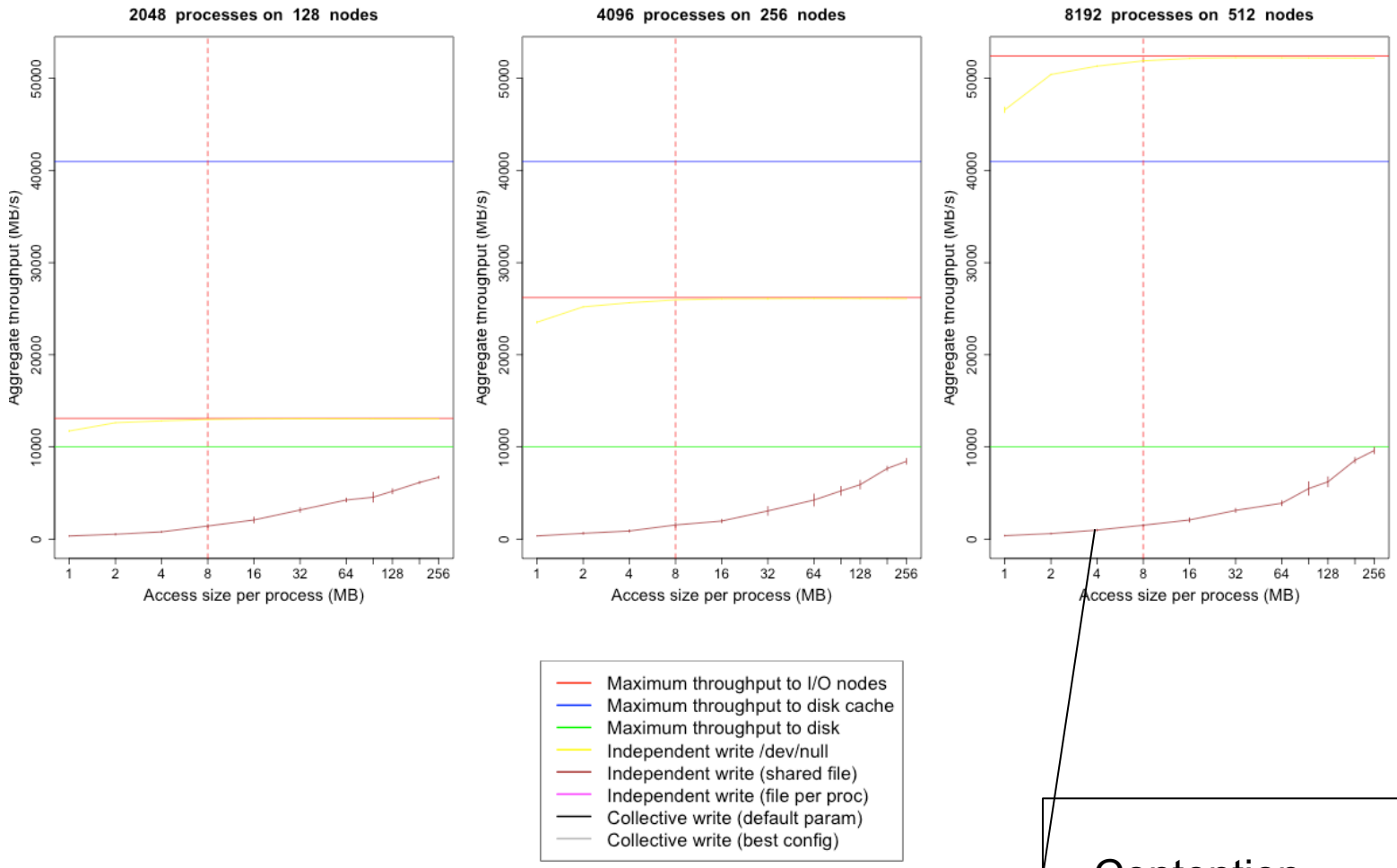
Aggregate write throughput on BG/Q



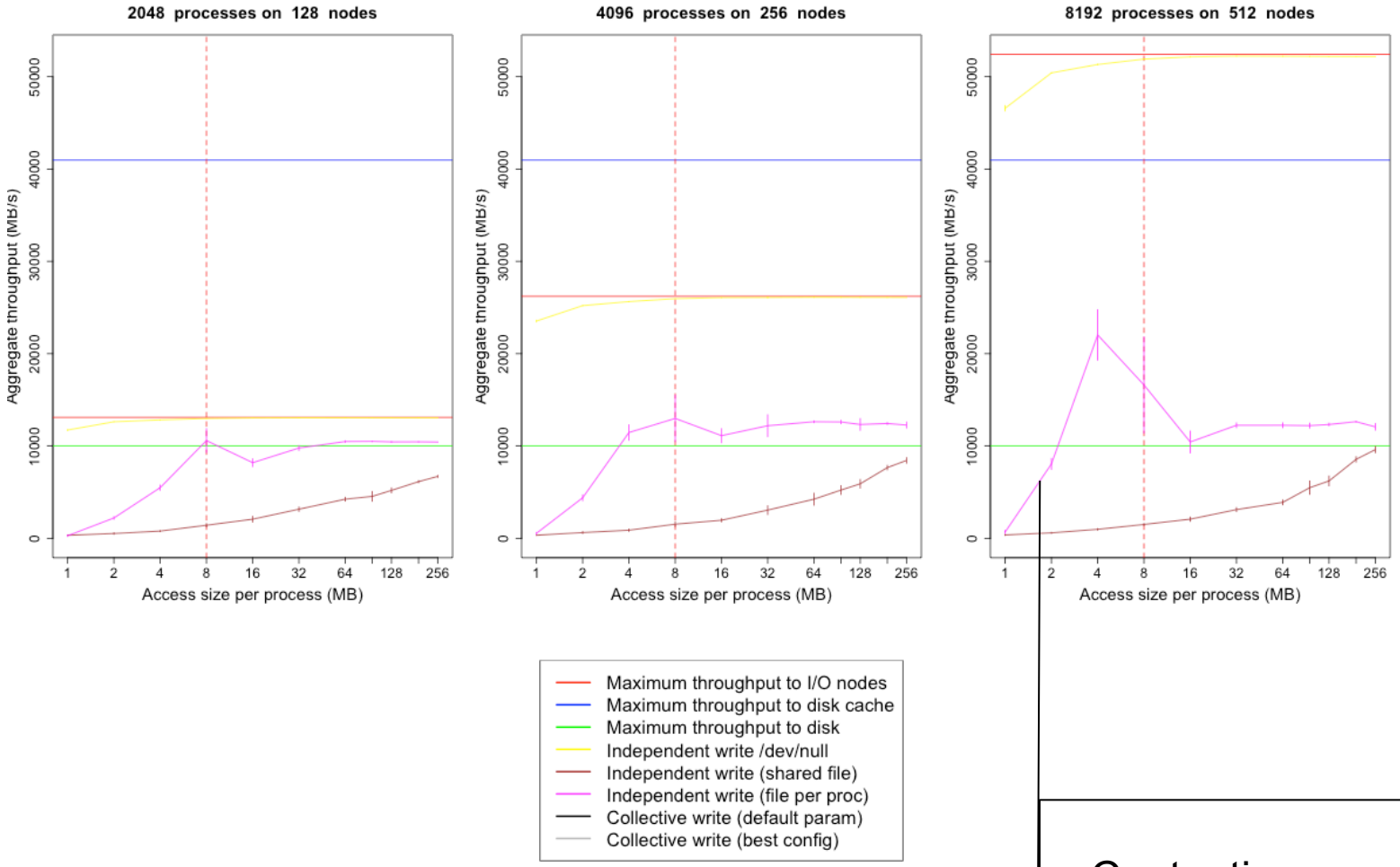
Aggregate write throughput on BG/Q



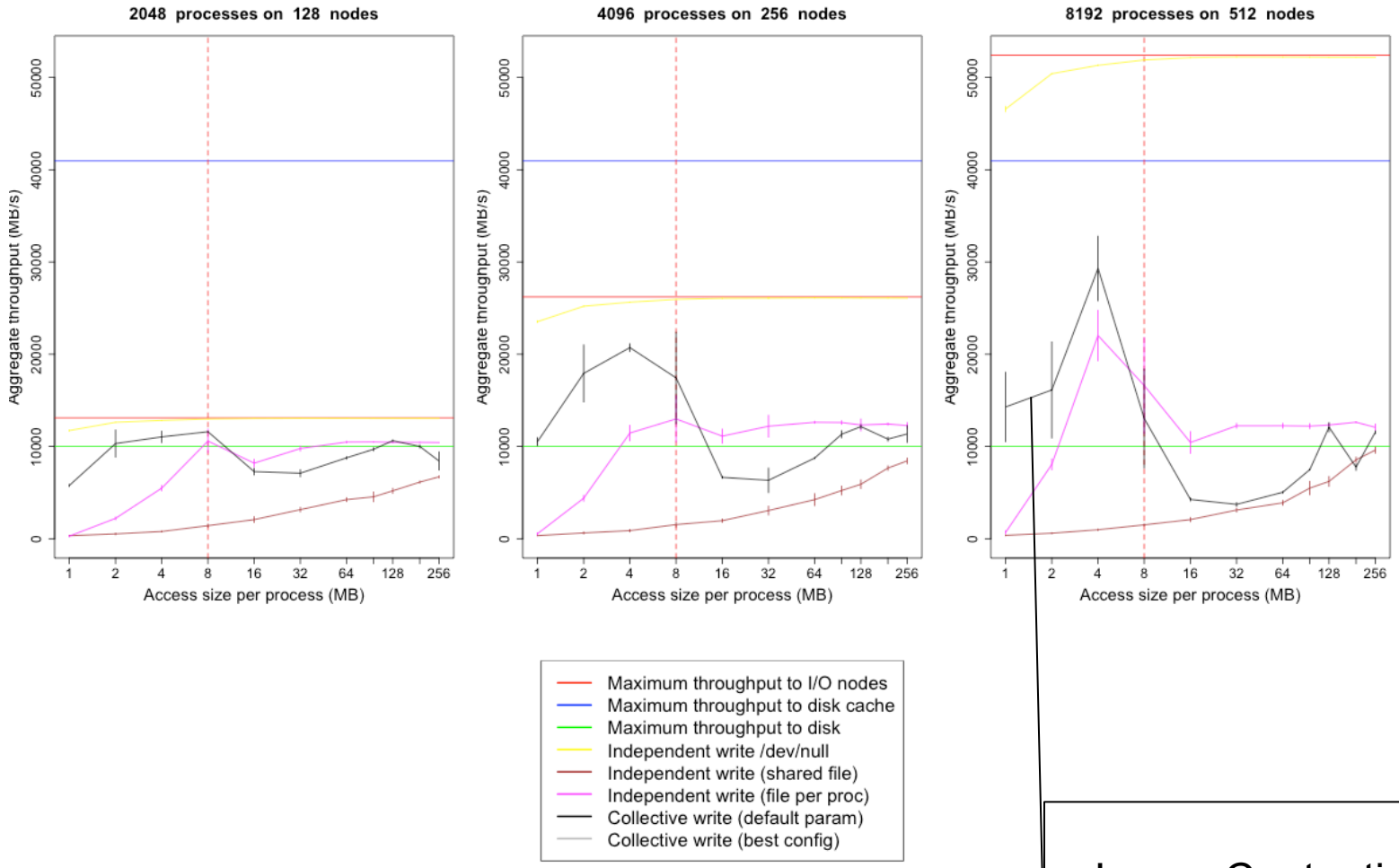
Aggregate write throughput on BG/Q



Aggregate write throughput on BG/Q

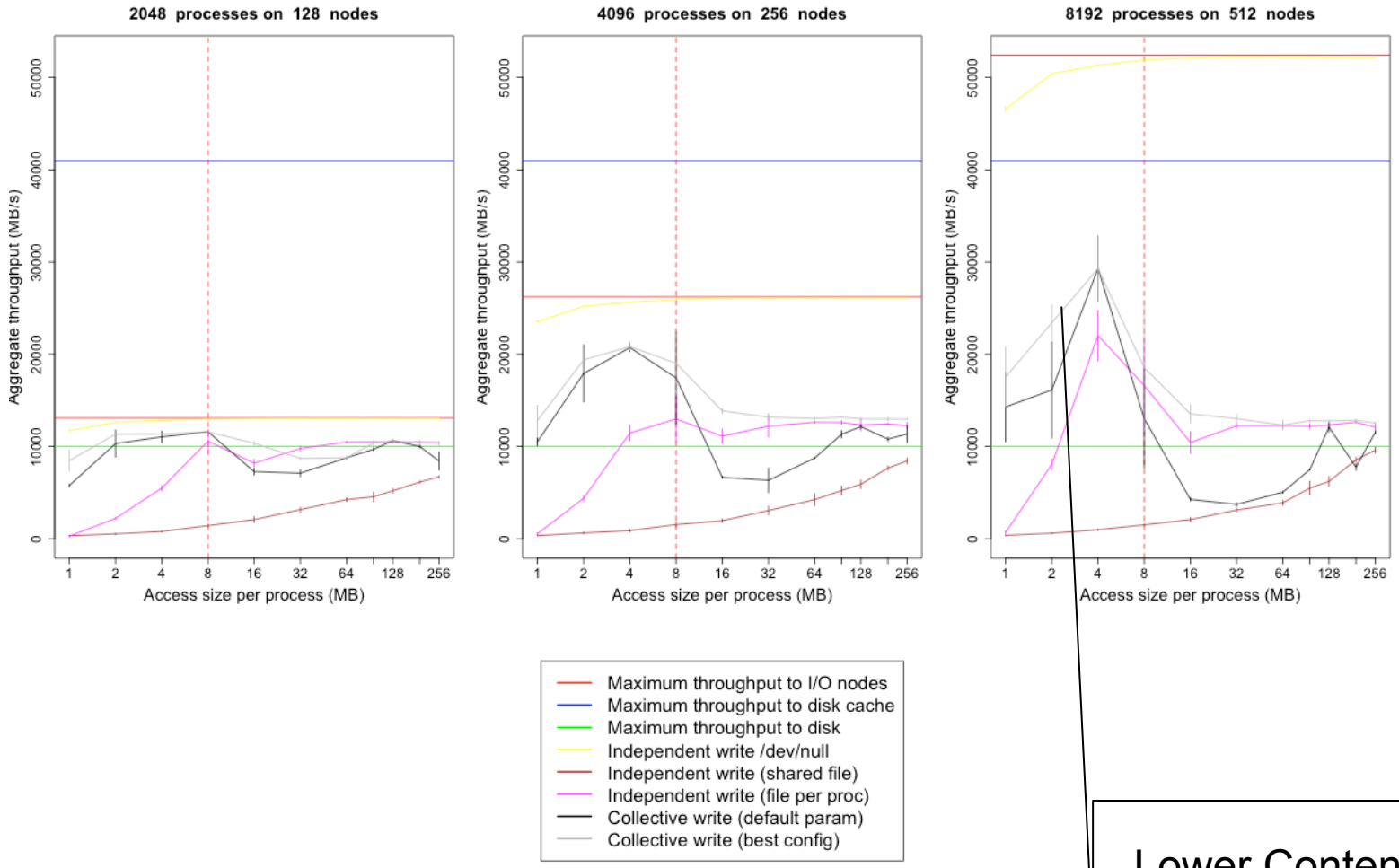


Aggregate write throughput on BG/Q



Lower Contention
GFS locking cost

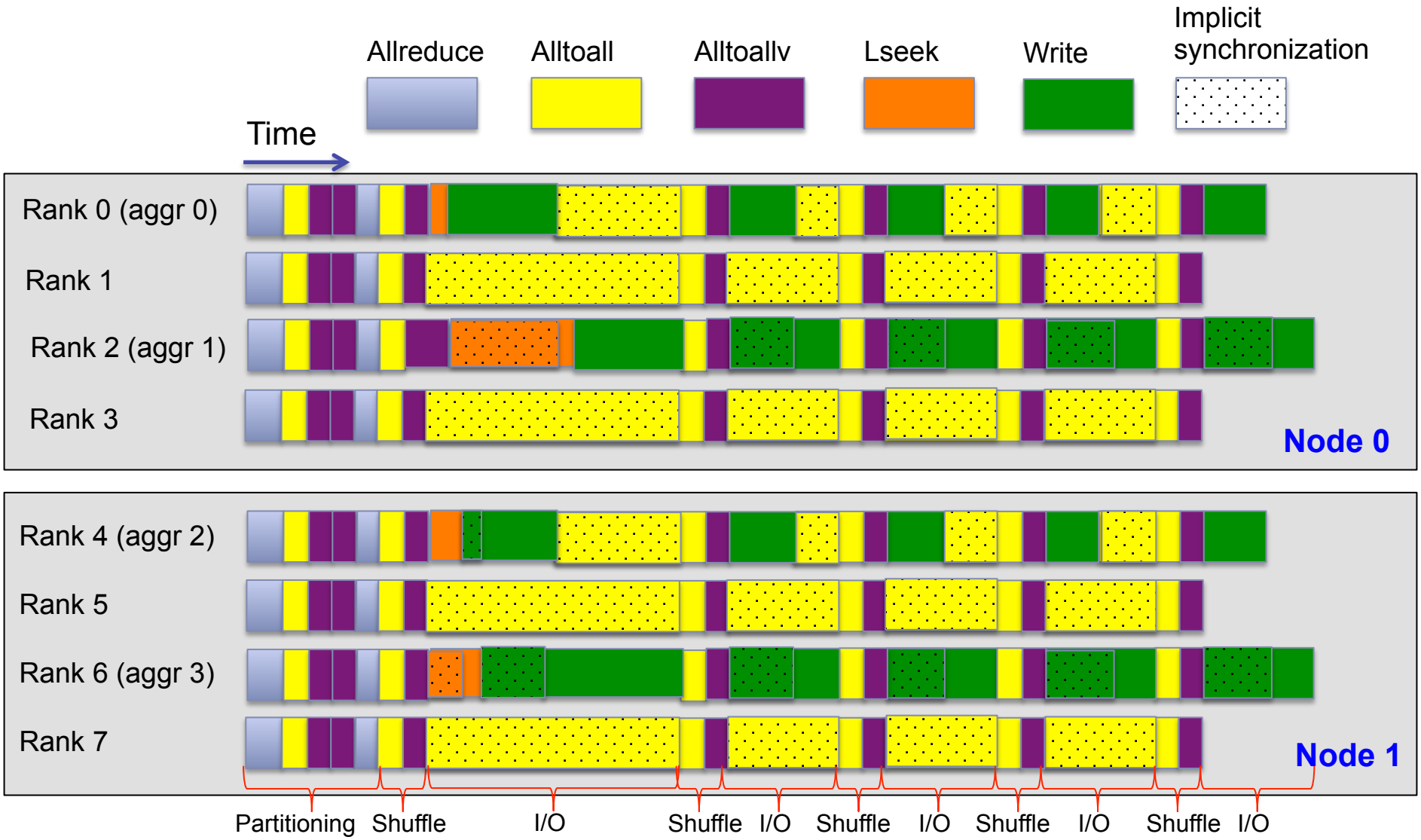
Aggregate write throughput on BG/Q



- Collective buffer size: 8MB, 16MB (default), 32MB
- Aggregators per 128 nodes: 40, 136, 520 (default)

Lower Contention
GFS locking cost
Lower implicit synch.

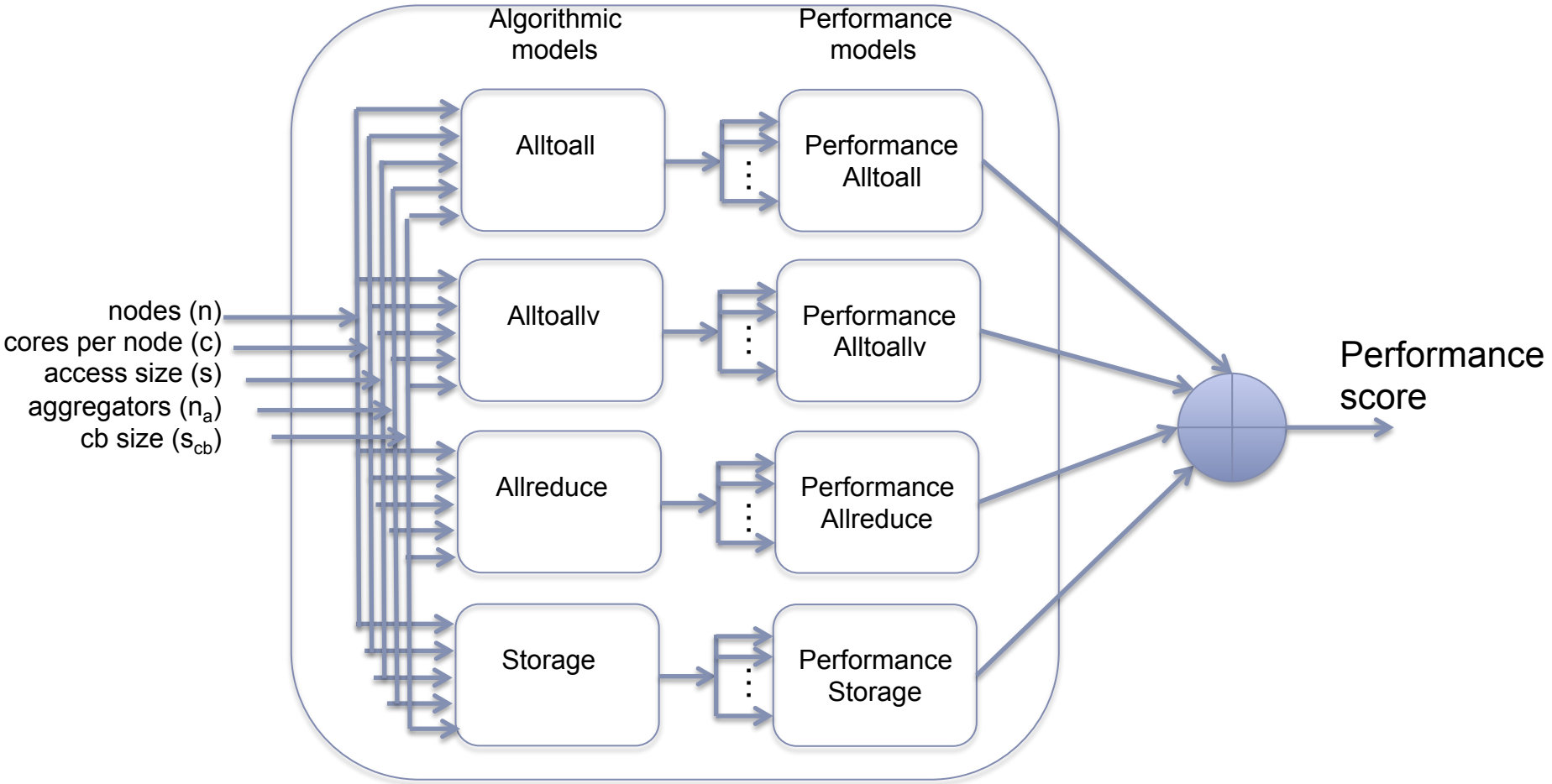
Anatomy of collective I/O



Model for ROMIO Collective I/O

- 5 parameters
  - n: number of nodes
  - c: cores per node
  - s: access size
  - n_a: number of aggregators
  - s_{cb}: collective buffer size
- For contiguous accesses closed form expressions for network and storage activity
  - Number of operations
  - Transfer size
- Goal: predict n_a and s_{cb}

Modeling framework



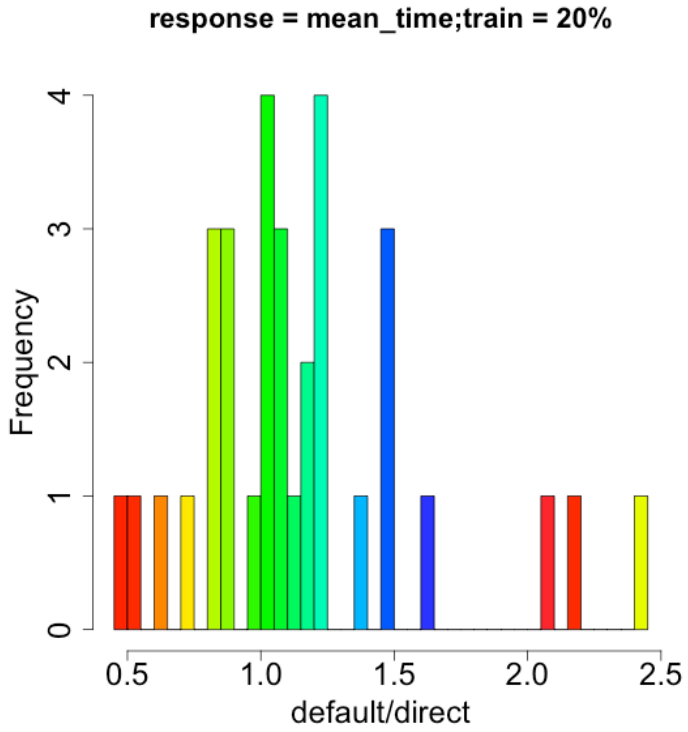
Network and storage performance

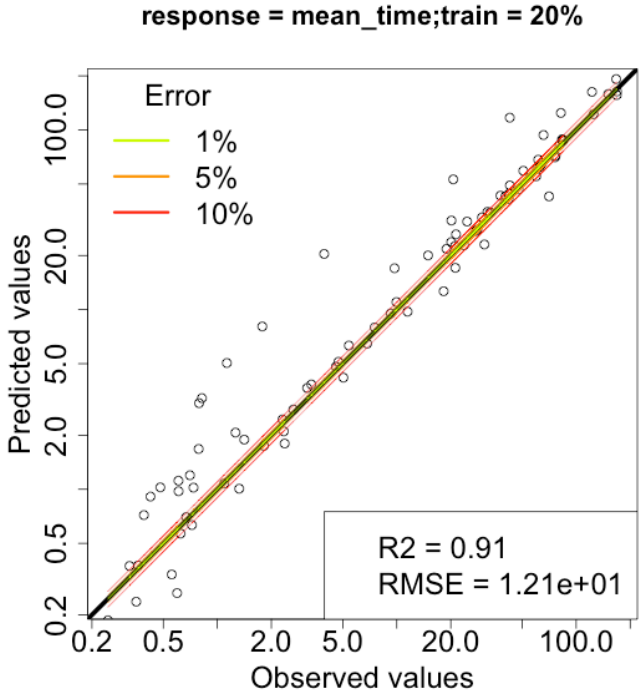
- Given a predictable network and storage performance, it is possible to exactly predict n_a and s_{cb}
- Network performance
  - On BG/Q stable performance
  - Implicit synchronization due to reuse of the collective buffer
    - exclusively due to storage I/O
- Storage performance
  - Storage hierarchy: compute node -> I/O node -> disk caches -> disks
  - Noise due to other applications accessing the file system
  - I/O forwarding layer implementation: calls forwarded from the cores of a compute node are serialized (e.g. a lseek arriving shortly after a write waits until the write is forwarded)
  - Concurrency: parallel access from several aggregators
  - POSIX consistency semantics for sharing the file

Statistical model

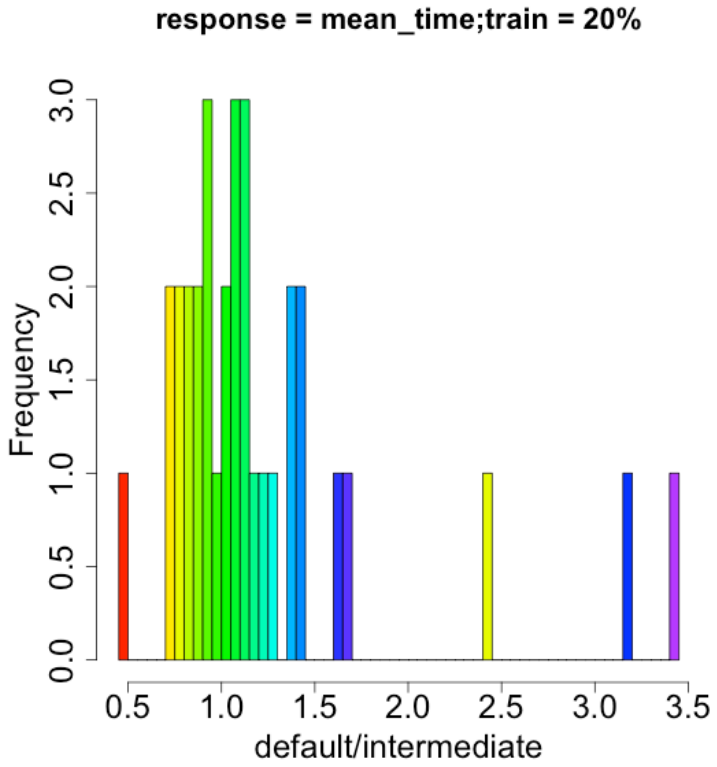
- Data set:
  - Nodes: 128, 256, 512
  - Cores per node: 16
  - Transfer sizes/core (MB): 1, 2, 4, 8,16, 32, 64, 96, 128, 192, 256
  - Collective buffer size (MB): 8, 16 (default), 32
  - Number of aggregators per 128 nodes: 40, 136, 520 (default)
  - 3 x 1 x 11 x 3 x 3 = 297 data points
- Pure statistical model
  - Boosted regression trees (cubist)
  - 20% training set
  - 80% testing set (including the optimal parameters)
- Mixed model
  - Analytical models for predicting the number of operations and operation size
  - Statistical models for the performance of individual operations

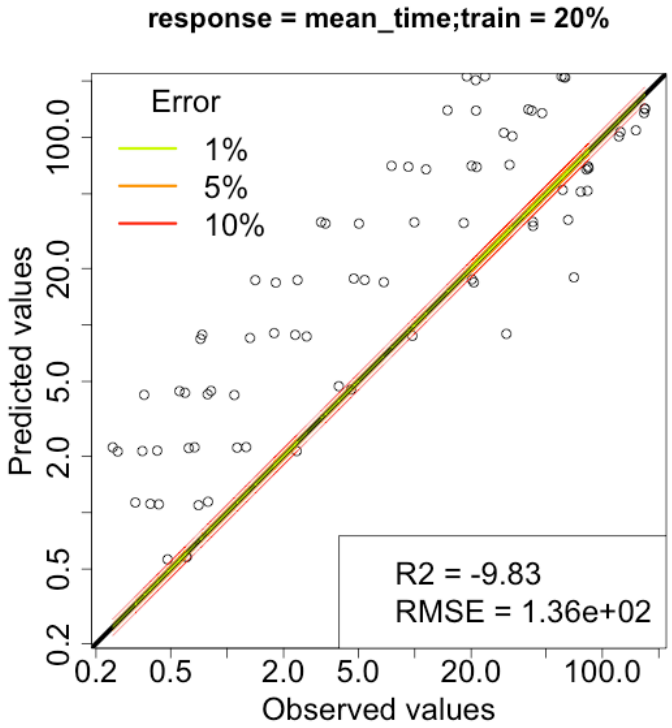
Pure statistical model





Analytical/statistical model





Conclusions

- Automatic parameter configuration
  - Black box modeling offers a limited benefit
  - Mixed model: Unstable performance of storage system a great hurdle
    - Having the complete knowledge of number of operations and operation size
    - But same operation radically different
- Factors that limit efficiency of the I/O stack optimization
  - Serialization in the I/O forwarding layer
  - File system noise
  - Implicit synchronization
  - POSIX consistency semantics
  - The lack of information about the state of other stack layers

Conclusions

- What is needed?
  - Better I/O scheduling
    - De-serialization of concurrent independent operations
    - Aggregation
  - Asynchrony (throughout the data stack and storage hierarchy)
  - Burst buffers (probably on I/O nodes)
  - An adequate consistency semantics for HPC (get rid of POSIX)
  - Mechanisms for facilitating the global reasoning about the optimization process

Thank you