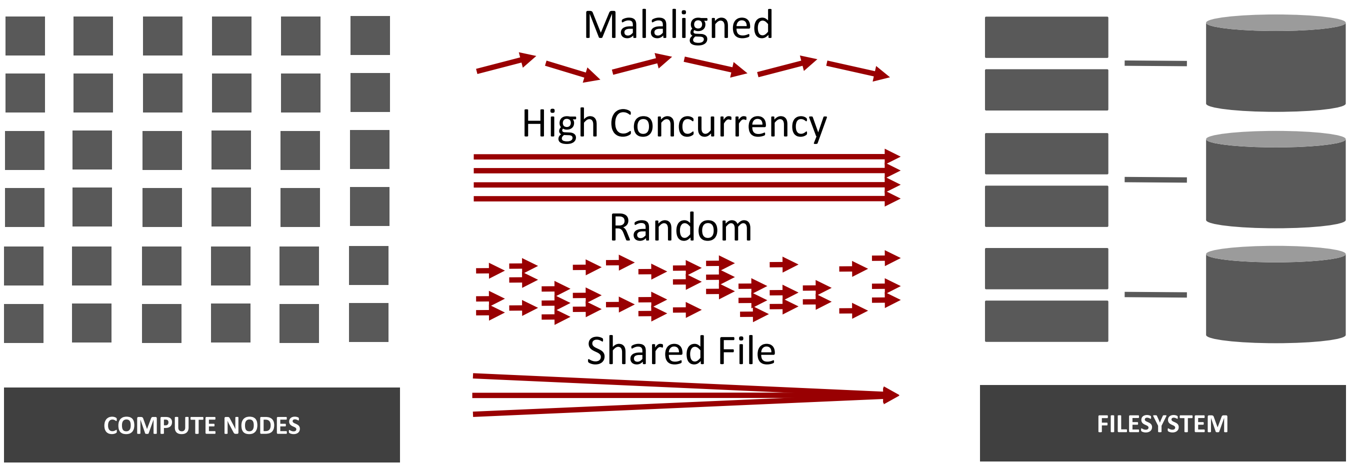
Infinite Memory Engine
Freedom from Filesystem Foibles

James Coomer
25th Sept 2017

Bad stuff can happen to filesystems



And Applications want to do bad stuff

Multi-Physics
Complex mixed I/O

Workflows+Ensembles
Globally coordinated I/O

Machine Learning
Read intensive

Adaptive Mesh Refinement
Varying I/O sizes

Checkpoints
Shared File I/O

High Concurrency
Extreme thread counts

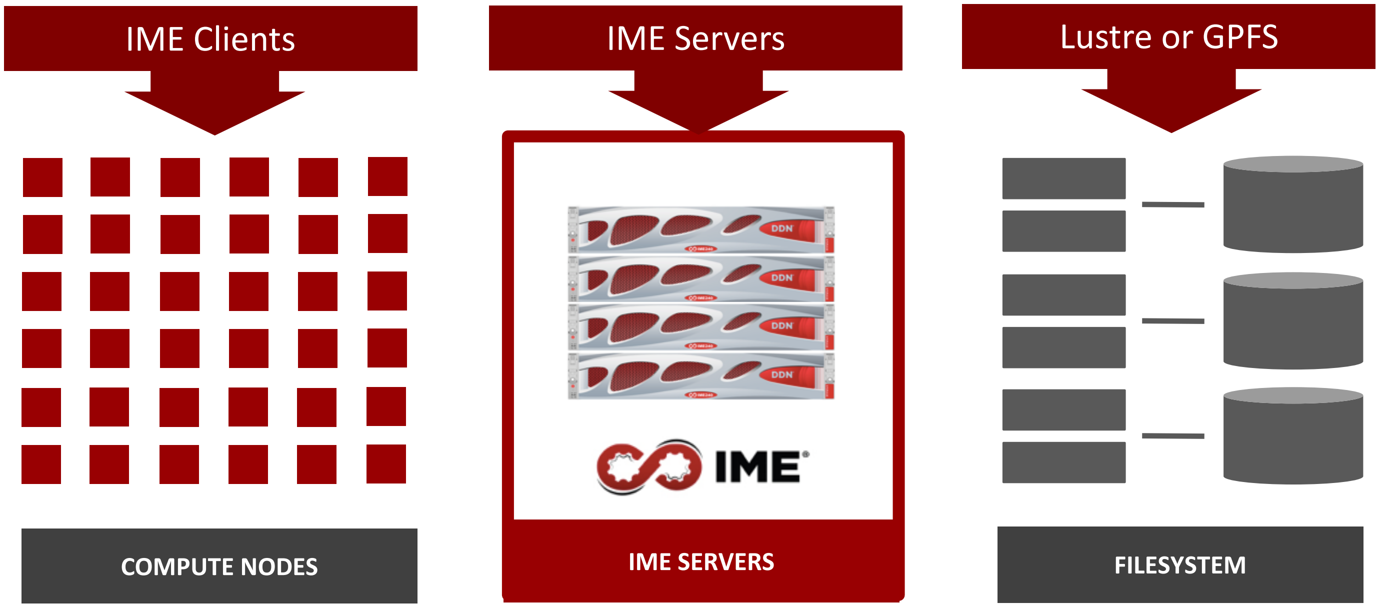
Free your applications from the limits of filesystems

How do I make my application go faster?

- Optimal/Natural behaviour already exist within the algorithms
- Difficult to optimise for multiple aspects simultaneously (GPU, multi-threading, network changes, caches)
- Difficult to optimise our application for new environments
- Difficult to maintain efficient porting even between different node counts

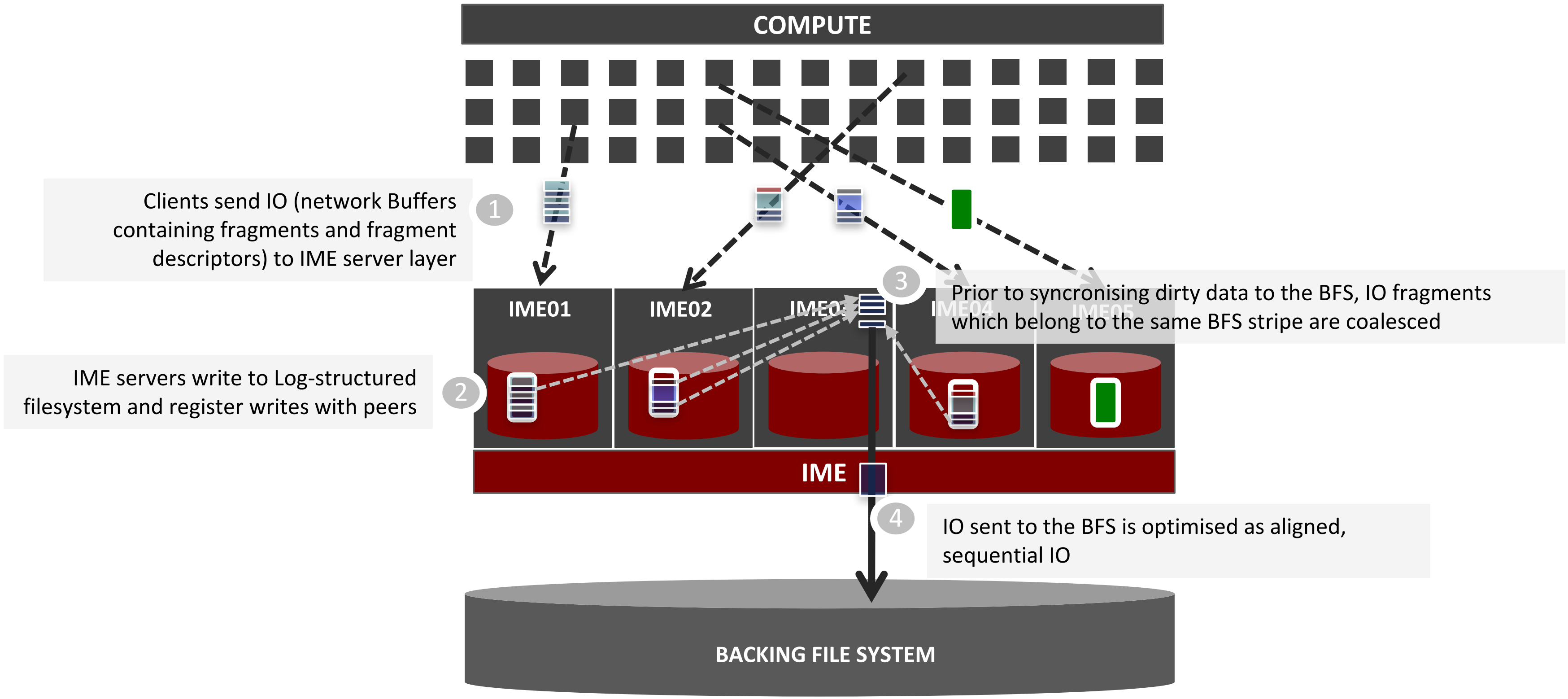
How do I make my application go faster?

IME | Scale-Out Data Platform



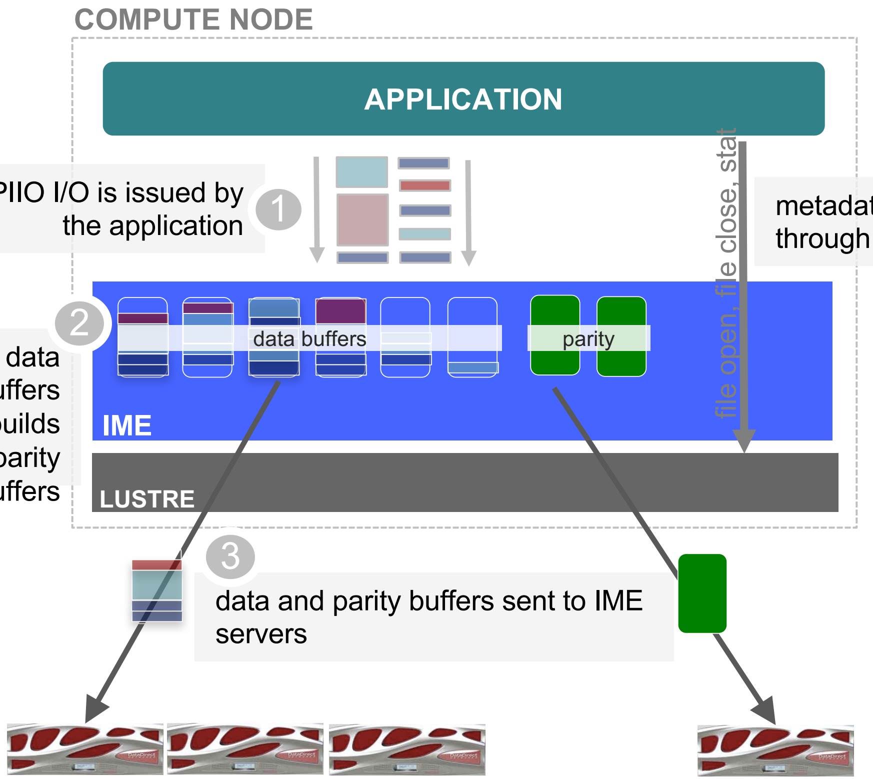
IO Management

Data Flow to Servers



IO Management

DataFlow in the Client: Flexible Erasure Coding



POSIX or MPIIO I/O is issued by the application

IME places data fragments into buffers and builds corresponding parity buffers

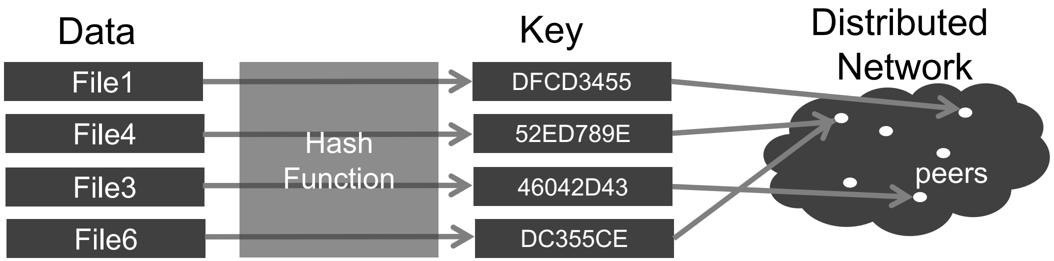
metadata requests passed through to the PFS client

data and parity buffers sent to IME servers

Erasure coding options:

1+1	1+2	1+3
2+1	2+2	2+3
2+3	3+2	3+3
...	...	...
15+1	15+2	15+3

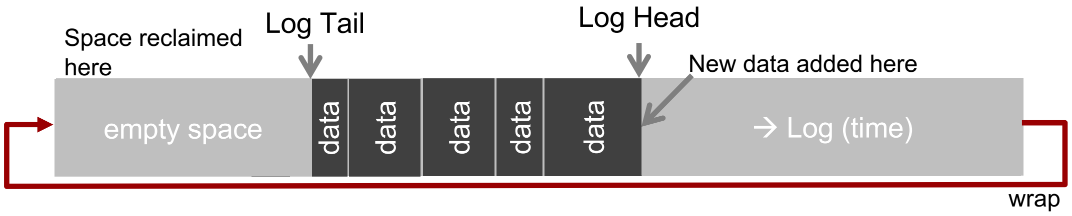
DISTRIBUTED HASH TABLE



DHT provides foundation for

- Network parallelism
- Node-level fault tolerance
- Distributed metadata
- Self-Optimising for Noisy Fabrics

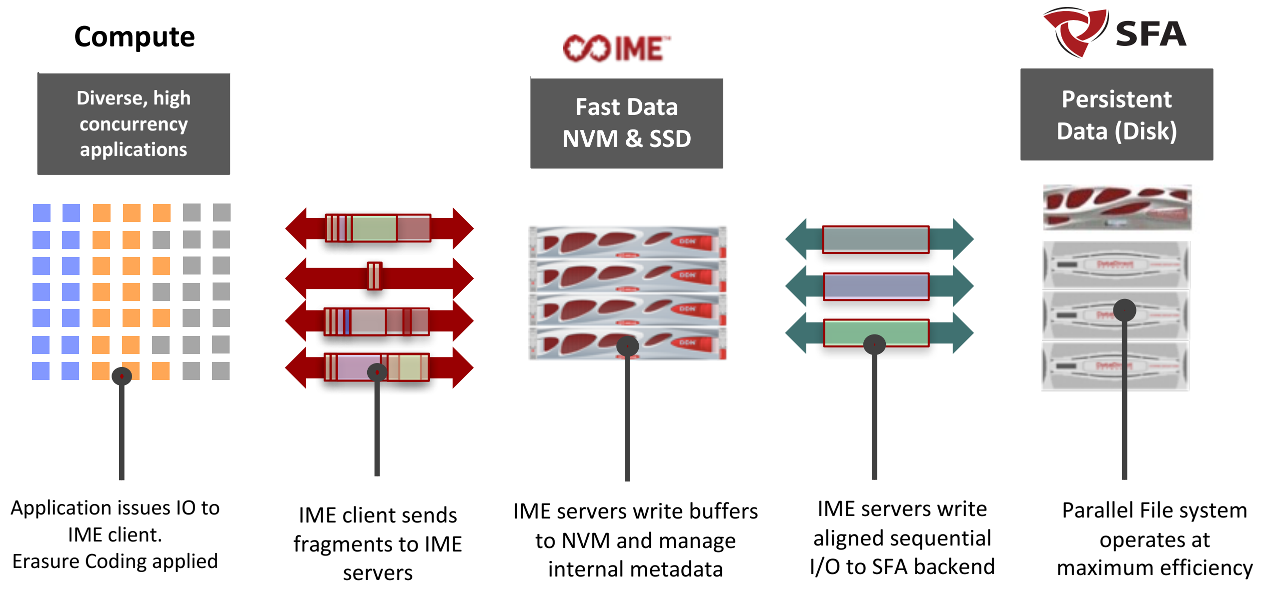
LOG STRUCTURED FILESYSTEM



Log Structured Filesystem at the storage device level

- High performance device throughput (NAND Flash)
- Maximises device lifetime

IME DataFlow



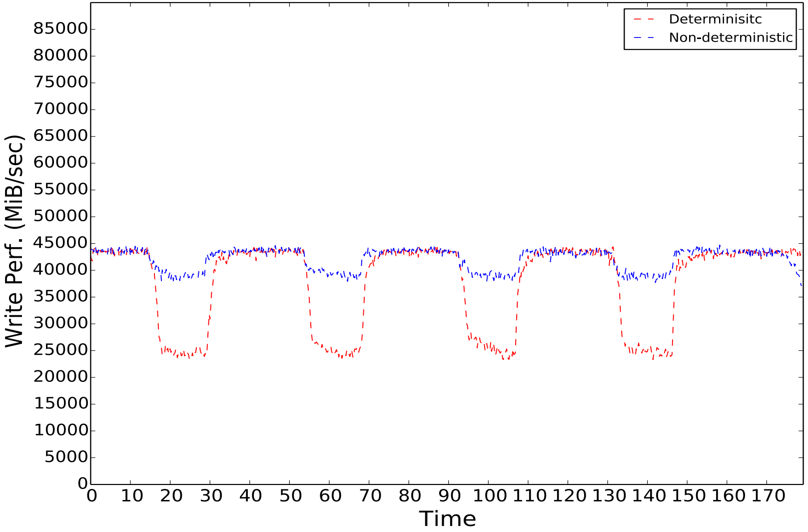
Amdahl's Law applied to parallel filesystems

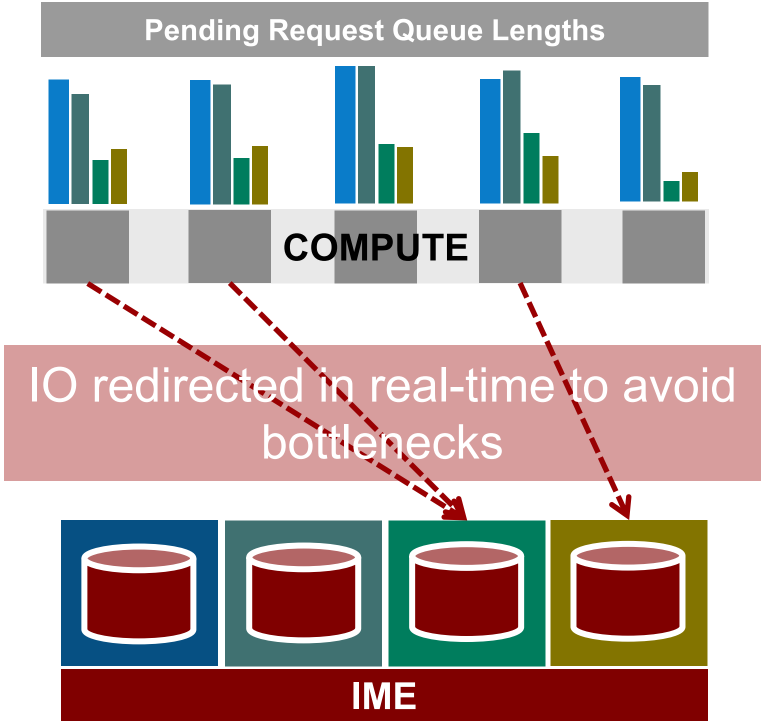
The system will run as fast as the slowest component

Fabric-Aware

Self-Optimising IO

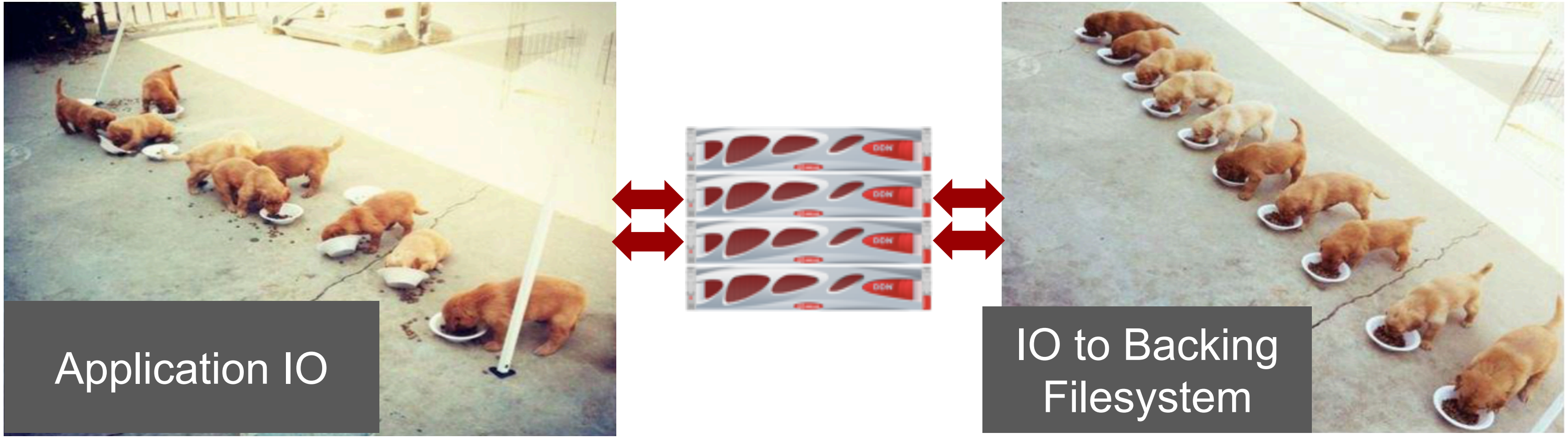
- IME clients "learn" and observe the load of IME servers and route write requests to avoid highly-loaded servers



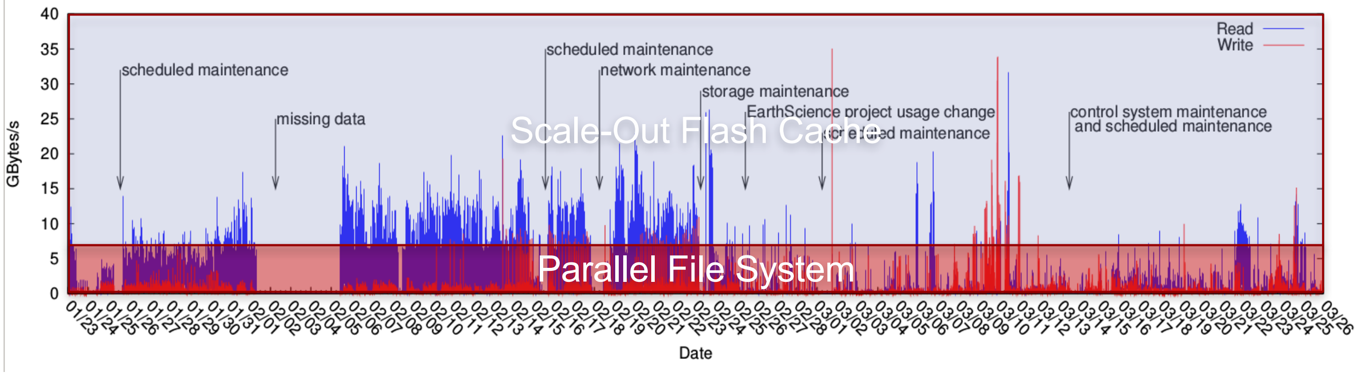


IME IO Deblender

improve efficiency of Backing FIlesystem



Flash-Cache Ecomonics

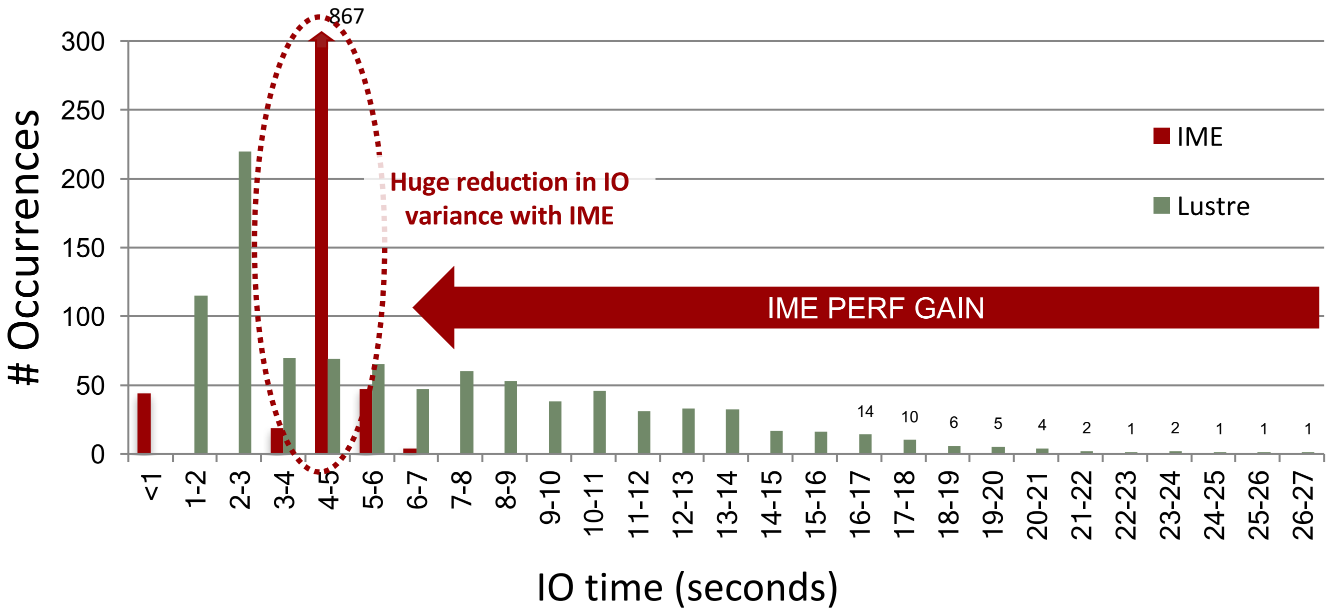


- **Real World IO sub-system activity**
  - 99% of the time <30%
  - 70% of the time <5%
- **→ Build a cache for the peaks, PFS for the capacity**
- **10x Performance/Watt improvement for flash-cache over PFS**

Argonne: P. Carns, K. Harms et al., Understanding and Improving Computational Science Storage Access through Continuous Characterization

IME | predictable performance

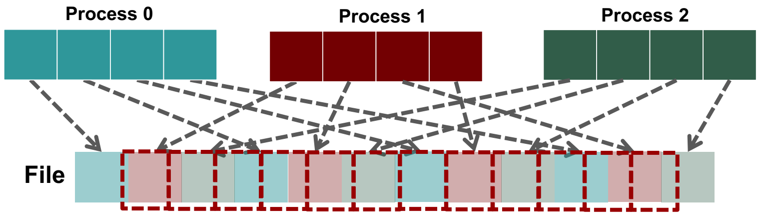
Eliminate IO commit variance of parallel file system

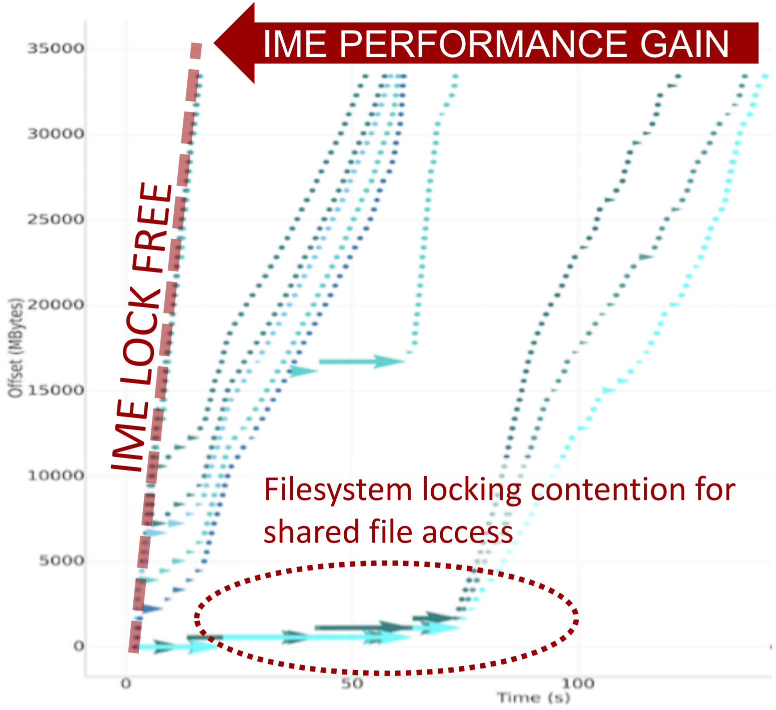


Results taken for large scale WRF benchmarking

IME | Shared File IO improvements

- Parallel File systems can exhibit extremely poor performance for shared file IO due to internal lock management as a result of managing files in large lock units
- IME eliminates contention by managing IO fragments directly, and coalescing IO's prior to flushing to the parallel file system





Use of Log Structuring in IME

Consider two different application I/O patterns (write)

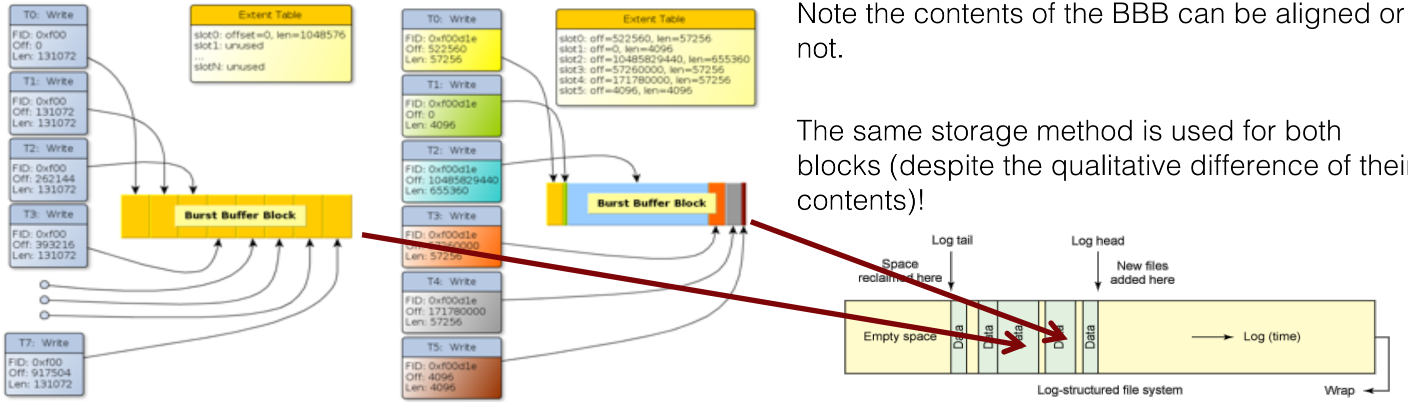
Sequential

Non-sequential

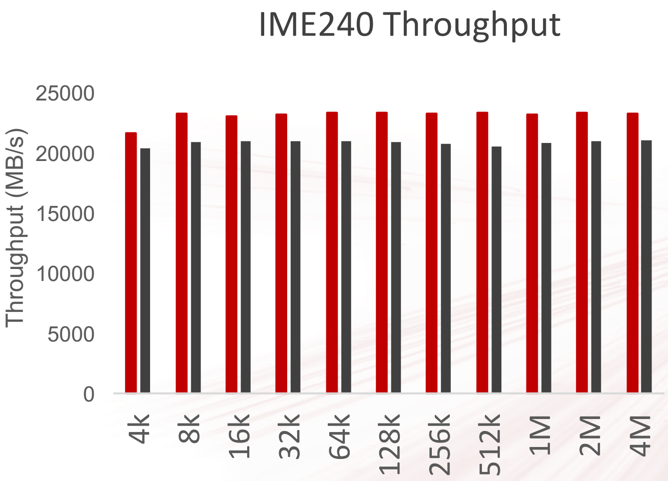
'*Burst buffer blocks*' (BBB) are really just buffers generated at the client

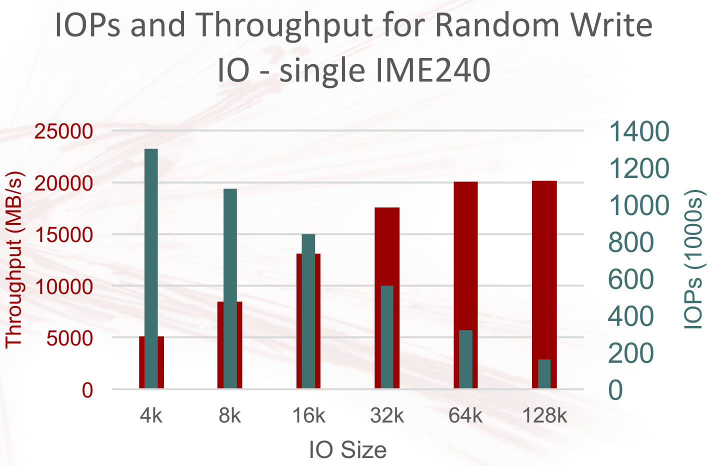
Note the contents of the BBB can be aligned or not.

The same storage method is used for both blocks (despite the qualitative difference of their contents)!



IME | Straight Line Performance





>1M IOPs | >20GB/s | 2 Rack Units

Thank You!

Keep in touch with us

sales@ddn.com

@ddn_limitless

company/datadirect-networks

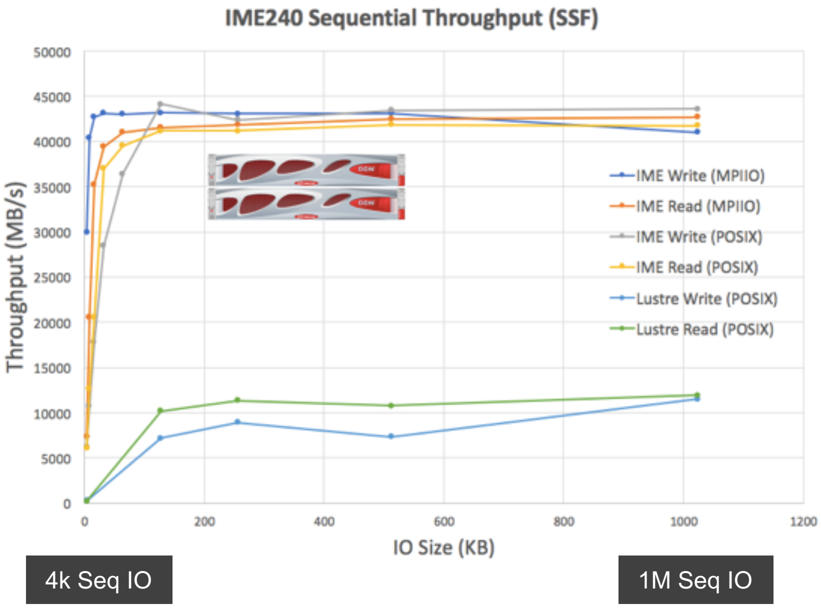
9351 Deering Avenue, Chatsworth,
CA 91311

1.800.837.2298
1.818.700.4000

IME1.1 | Blistering Sequential Performance

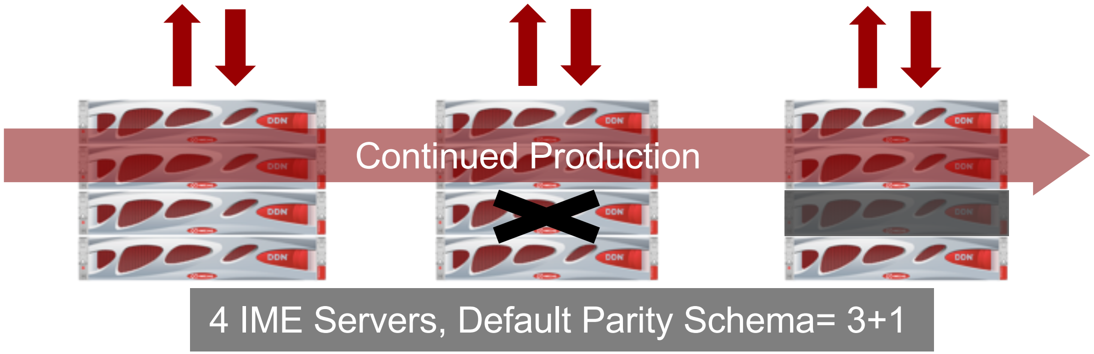
Single Shared File

- IME240 Sequential performance for Single Shared File over **22GB/s per server** (POSIX IO)
- 15GB/s with 4k IOs **(MPIIO)**
- **consistent read and write performance across IO sizes**
- **POSIX IO performance hits max values at ~32K IO sizes**



IME1.2 Node Failure Management

- IME1.1 allows continued IO Service through IME server failure(s)
- Following server failure, IME commences background rebuild of all missing extents for each Parity Group Instance (PGI) : 3+1 across the 3 remaining servers
  - 1 server will be overloaded with 2 blocks for each PGI
  - Following a server failure but prior to rebuild completion, immediate read requests may be satisfied through on-the-fly reconstruction of missing file extents
- Service interruption will occur with a second IME server failure in *this* case (but not with NVM device failures that occur after rebuild)
- New writes still maintain 3+1 (with overloading)
- IME restart is needed to re-introduce failed node and restore original redundancy. (requires full sync to the BFS)



IME | HPC IO in the Flash Era

Adaptive IO

IME Clients adapt IO rates to the server layer according to load eliminating traditional IO system slowdowns

Dial-in Resilience

Erasure coding levels are not dictated by storage system setup, but dynamically set on a per file/per client basis

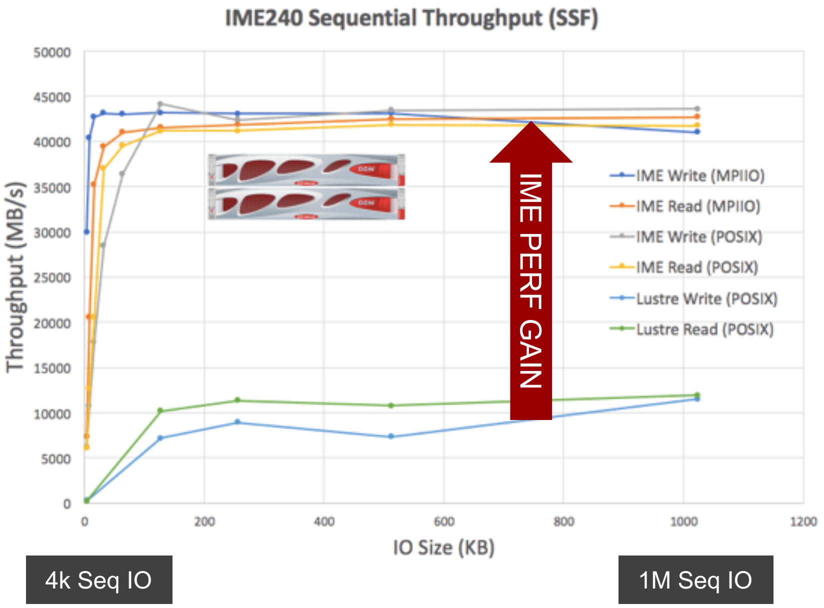
Lightning Rebuilds

Fully Declustered distributed Data rebuilds allow for rebuild rates in excess of 250GB/minute

IME | Shared File Performance

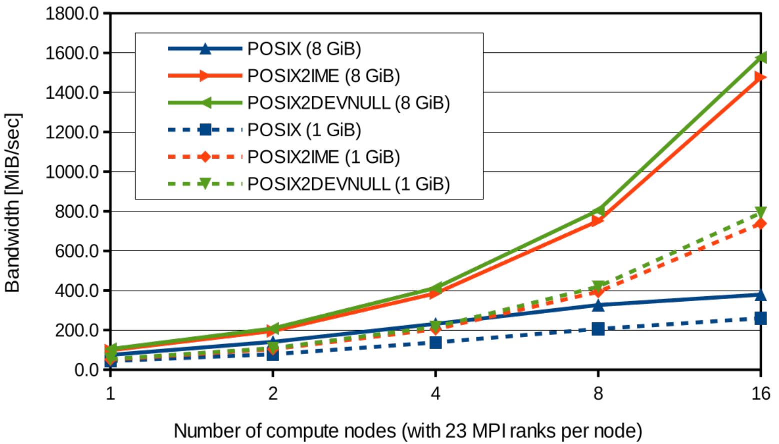
Shared File Performance Acceleration

- IME240 Sequential performance for Single Shared File over **22GB/s per server** (POSIX IO)
- 15GB/s with 4k IOs **(MPIIO)**
- **consistent read and write performance across IO sizes**
- **POSIX IO performance hits max values at ~32K IO sizes**



NEST | NEural Simulation Tool

- Dynamics of interactions between nerve cells
  → MPI + OpenMP
- I/O pattern burst of write at the GPFS scales imperfectly
  - 200 MB/s for 4 nodes
  - 400 MB/s for 16 nodes
- IME scales almost perfectly
  - 400 MB/s for 4 nodes
  - 1500 MB/s for 16 nodes



IME and /dev/null show nearly identical behavior.

Except IME keeps the data.

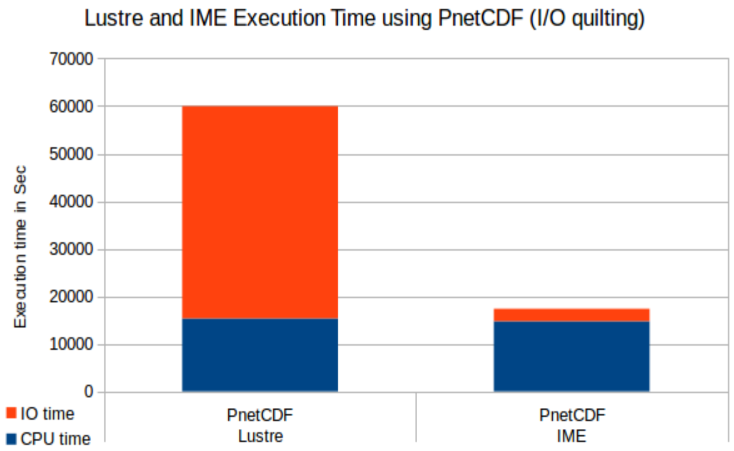
Courtesy of JÜLICH FORSCHUNGSZENTRUM

IME | WRF with I/O quilting

one I/O process per node

- **Application wall time reduced by 3.5 using IME**
- **I/O time reduces by x17.2**
- **Move from I/O bound to compute bound**

Measurement	Speed Up
IO Wallclock	17.2x
Total Execution Wallclock	3.5x

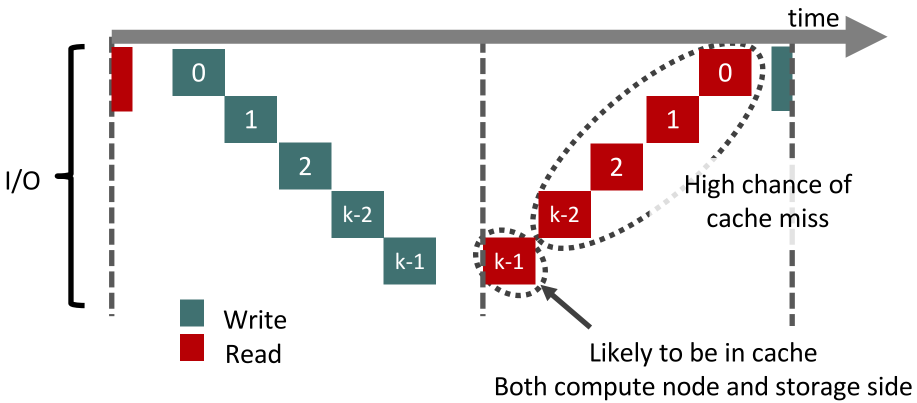


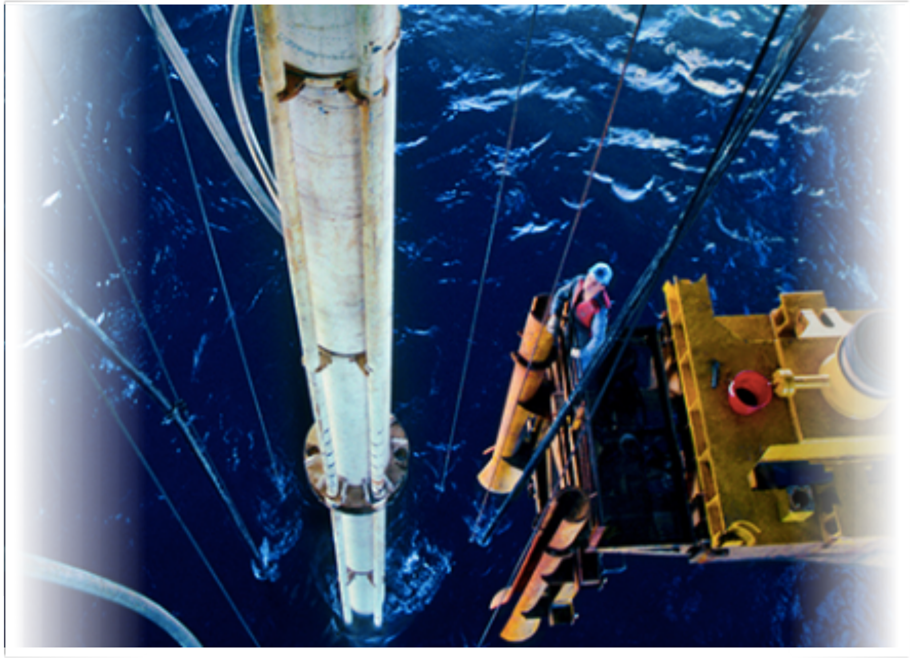
IME240 Starting Solution

- 3+1 Erasure Coding
- Raw Performance 80GB/s
- EC Perf: 60Write + 80Read
- Accelerate Existing Lustre and GPFS solutions
- Cope with Demanding Applications
- Relieve cross-contamination between filesystem users

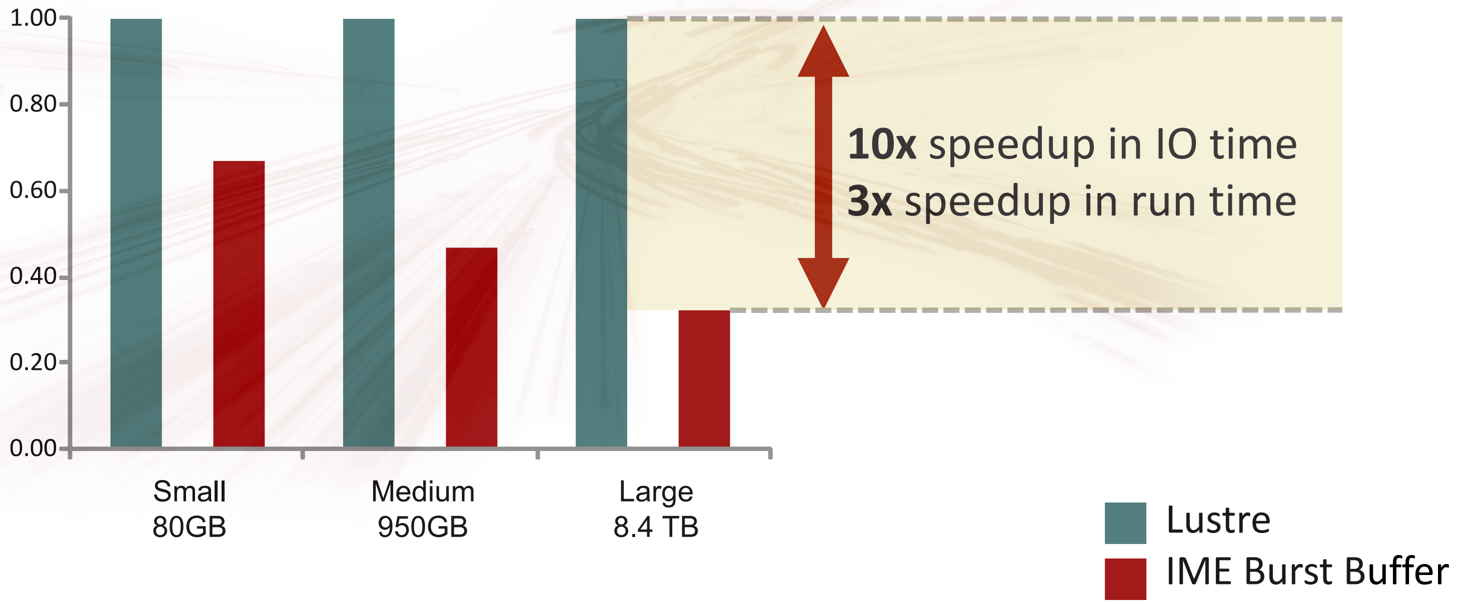
IME | Oil and Gas | Seismic

- IME expands the cache volume from GBs to PBs and eliminates cache misses associated with Reverse Time Migration IO patterns



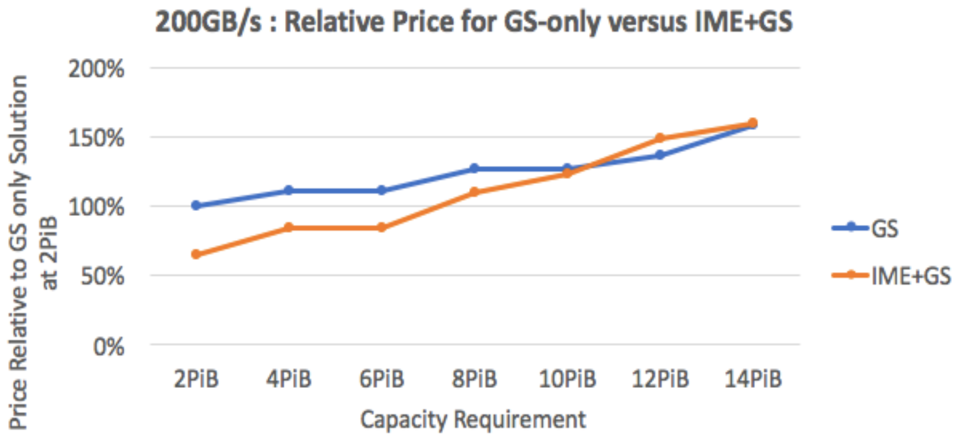


IME | TORTIA (Reverse Time Migration Code)



200GB/s example: IME flash-cache economics

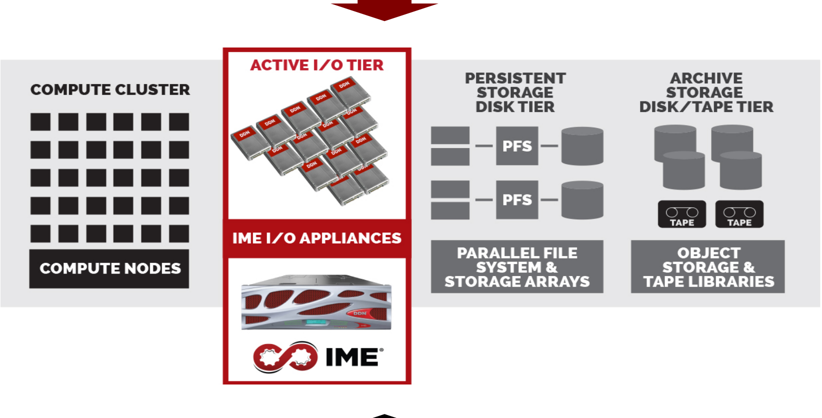
- For a 200GB/s system requirement, an IME solution price will be lower than that of an HDD Filesystem for capacities < 10PiB
- For an 8PiB Requirement:
  - Pure Filesystem: 6x GS14KX, 2400 4T drives
  - IME: 10x IME240, 1xGS14KX, 1440 8T drives <-- 20% lower cost,



What is IME Today?

- New Cache Layer using NVMe SSDs inserted between compute cluster and Parallel File System (PFS)
  - IME is configured as CLUSTER with multiple SSD servers
  - All compute nodes can access cache data on IME
- Accelerates "bad" IO on PFS
- Accelerates small IO or random IO by high IOPS due to SSD and IO management
  - PFS is pretty good at large sequential IO
- Can be configured as cache layer having huge IO bandwidth
  - eg. over 1TB/sec BW on JCAHPC Oakforest-PACS

IME's *Active I/O Tier*, is inserted right between Compute and the parallel file system



IME software intelligently virtualizes disparate NVMe SSDs into a single pool of shared memory that accelerates I/O, PFS & Applications

DDN | IME

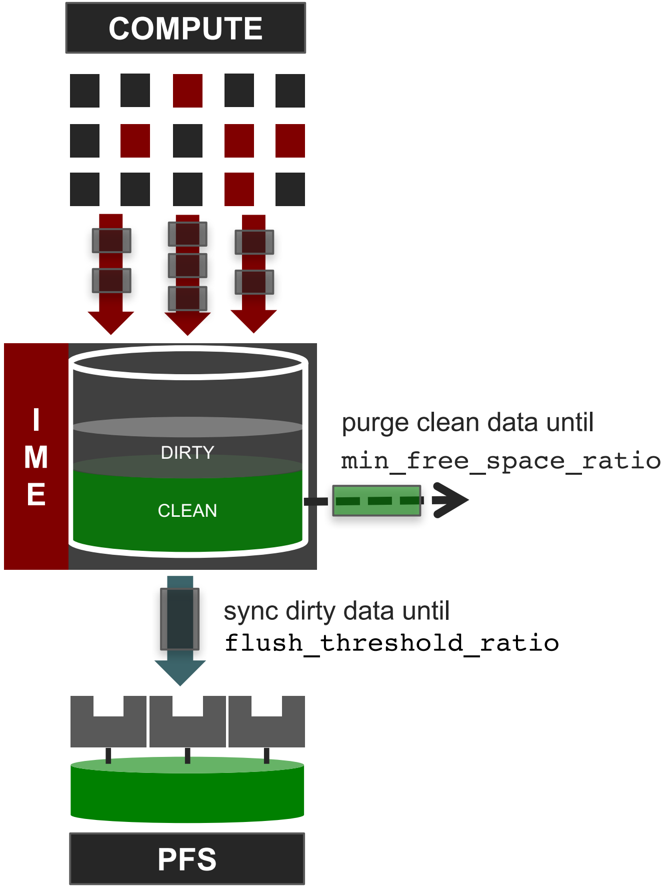
Data Residency Control

- maximum percentage of dirty data resident in IME before the data is automatically synchronized to the PFS:

```
flush_threshold_ratio   [0% .. 100%]
```

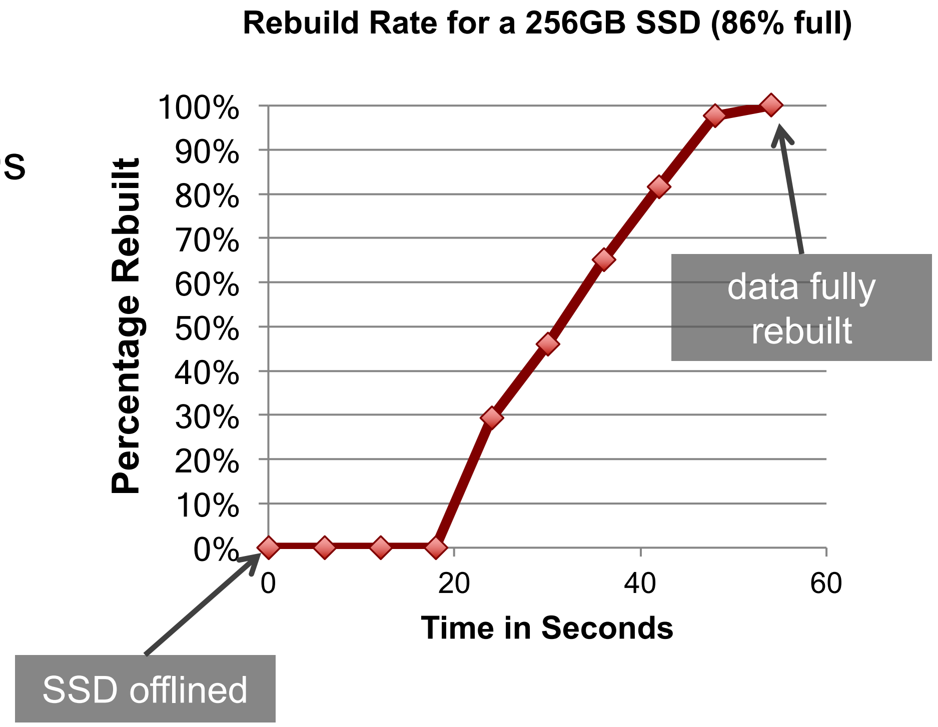
- Once Synchronised, the data is marked clean
- The clean data is kept in IME until the min_free_space_ratio is reached.

```
min_free_space_ratio    [0% .. 100%]
```



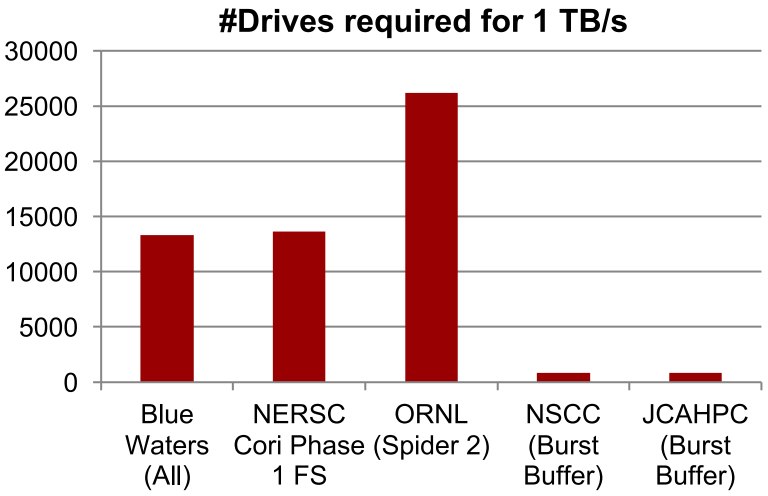
Extreme Rebuild Speeds

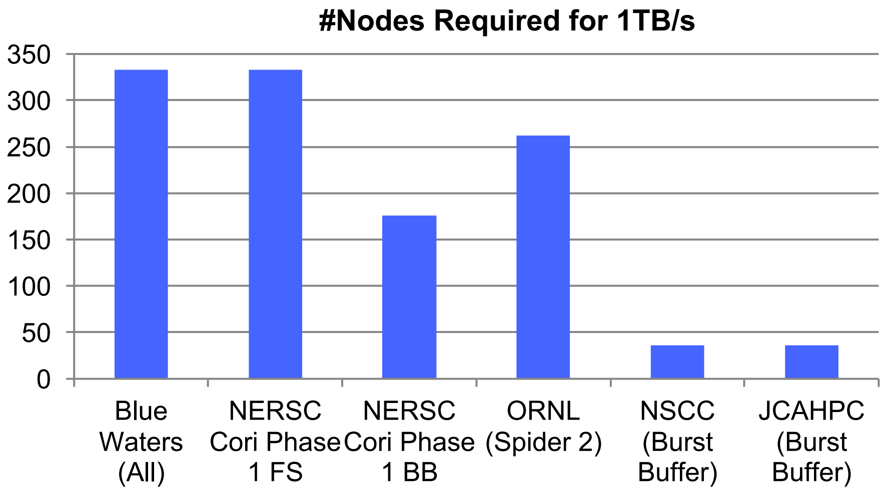
- Distributed Rebuild over all SSDs in Parity Group
- Each SSD performing rebuild at ~250MB/s
- 256GB SSD rebuilt in under 1 minute



New Ratios for Performance Systems

- IME removes restrictions of HDD-based capacity/performance ratios.
- Makes multiple TB/s manageable
- Dramatically reduces component count, power consumption, space consumption and capital cost. (see Astar model)

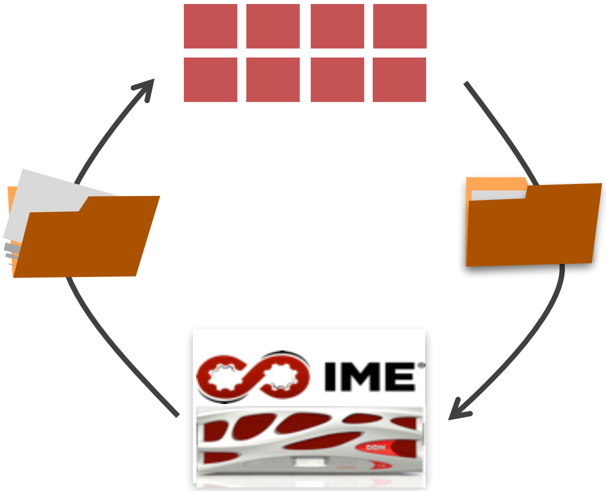




DDN | Data Consistency Model

Lock Free Model

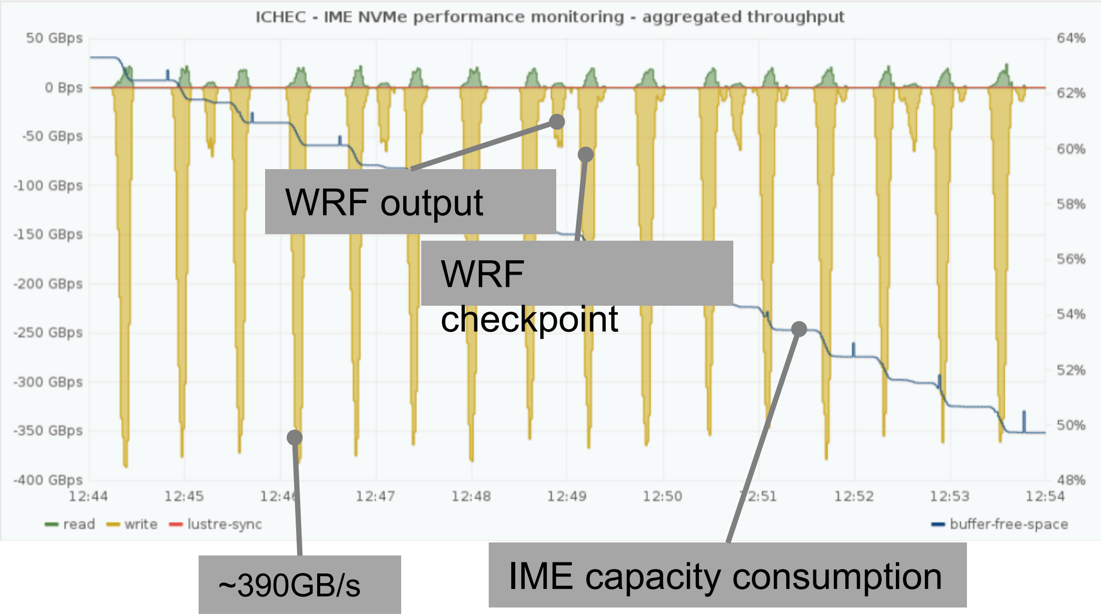
- Dirty or otherwise unsynchronized data within an IME Client's cache is not visible or retrievable by other clients.
- Synchronizing on close () ensures that IME Clients will provide close-to-open consistency

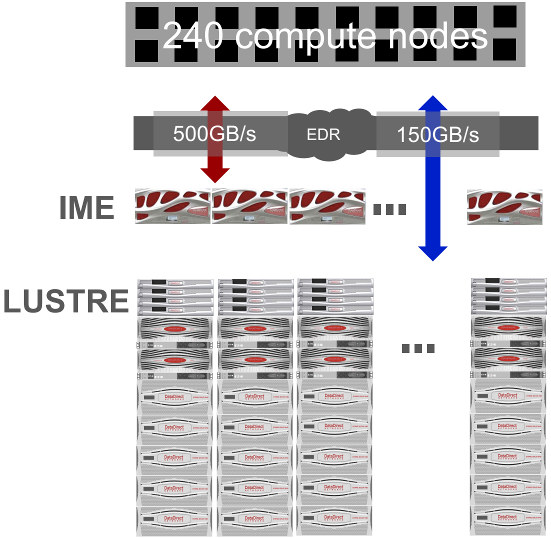


WRF on IME

48 jobs across 240 compute nodes

48 concurrent MPI job
5 node/job
20 MPI rank/node
IME erasure coding 7+1





WRF at Scale

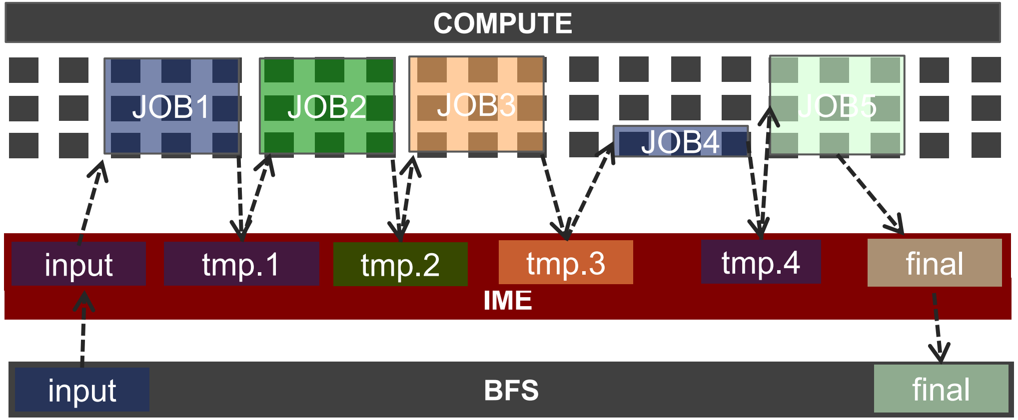
Summary Results

	IME		Parallel File system		
	#	Throughput per Metric (GB/s/<x>)	#	Throughput per Metric (GB/s/<x>)	**IME Improvement**
Application Throughput (GB/s)	380		100		**x 3.8**
Rack Units	36	10.5	224	0.45	**x 23**
# IO Nodes	18	21	42	2.4	**x 8.7**
# Drives	432	0.9	2800	0.04	**x 22**
Power Consumption (KW)	27	14	70	1.4	**x 10**

IME Use Cases

Jobs with Dependencies (workflows)

Application ensembles: Multiple, simultaneous applications use cache to support communication throughout the job.
Application workflows: cache stores a common dataset used by a succession of independent applications.
In-situ analysis: real-time output of an application's data in cache analysed in-situ
Application pre-processing: pre-processing job places output in cache, ready for main run.
Application post-processing: main run outputs to cache, post-processing commences in situ.
Visualization: users may want to keep the data available for use by shared compute systems.



- Benefit
  - Don't use PFS for scratch files
  - Fast IO for dependent jobs
- e.g. Weather
- visualisation environments