Infinite Memory Engine® (IME®): Accelerating Application and File Systems to make Clusters More Predictable

Steve Crusan – Systems Engineer
scrusan@ddn.com

Jan, 2018

2

Agenda

- IME Architecture Overview
- The effects of locking on a parallel file system
- Small scale test results, Lustre all flash vs IME (to show locking issues)
- Large scale results (IO-500)
- Questions
- Lunch

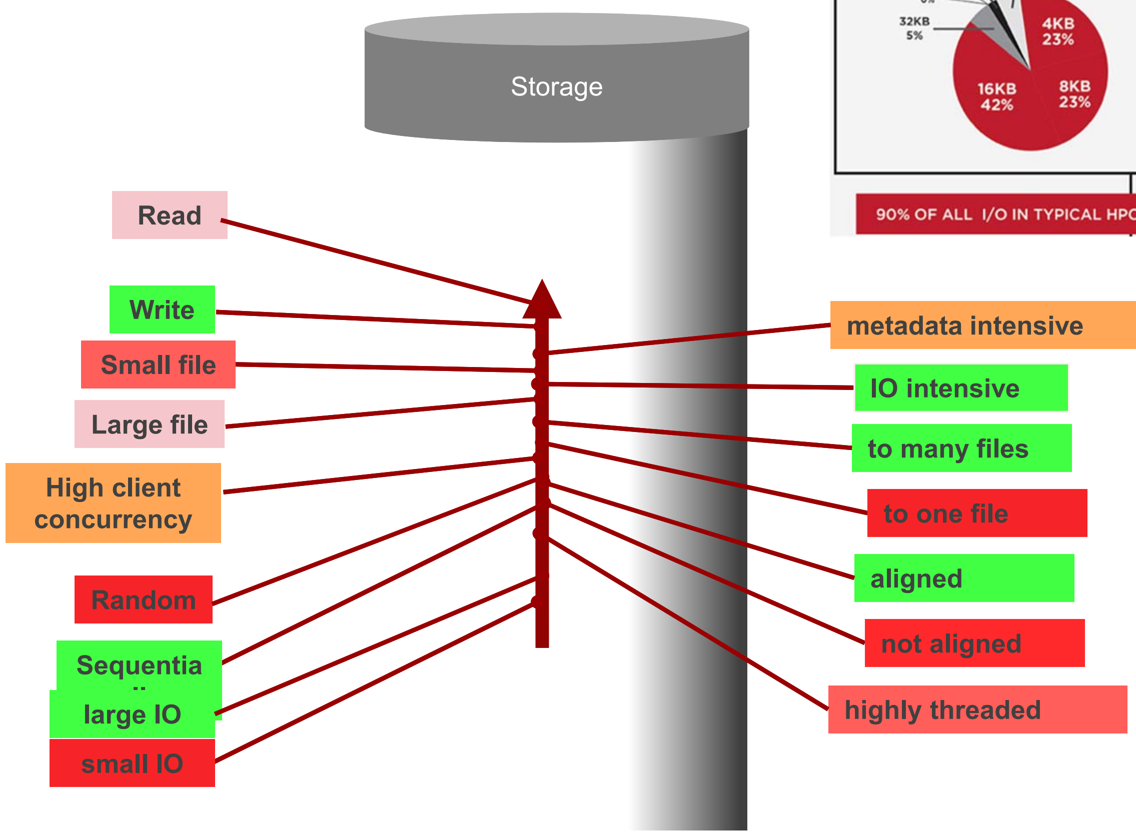
What is IME in 30 Seconds…

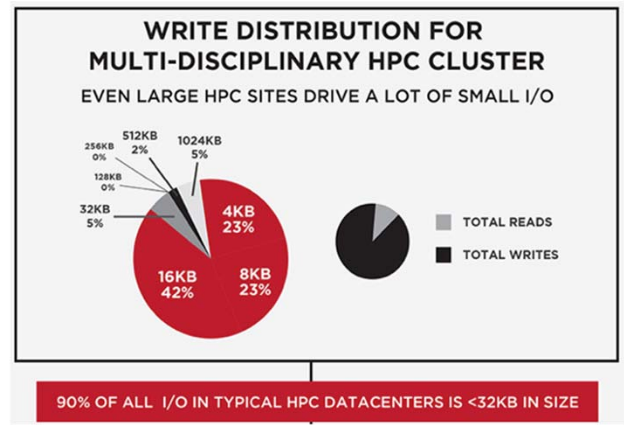
- Software defined, flash native I/O system
- Runs on commodity hardware, runs in the “cloud”
- More than a “burst buffer”, delivers real mixed, random, etc workload performance to the application. Only sequential I/O is boring in 2018.
- Scale out (largest system uses 8000 compute nodes)
- People make ridiculous logos with it:

4

IME Architecture

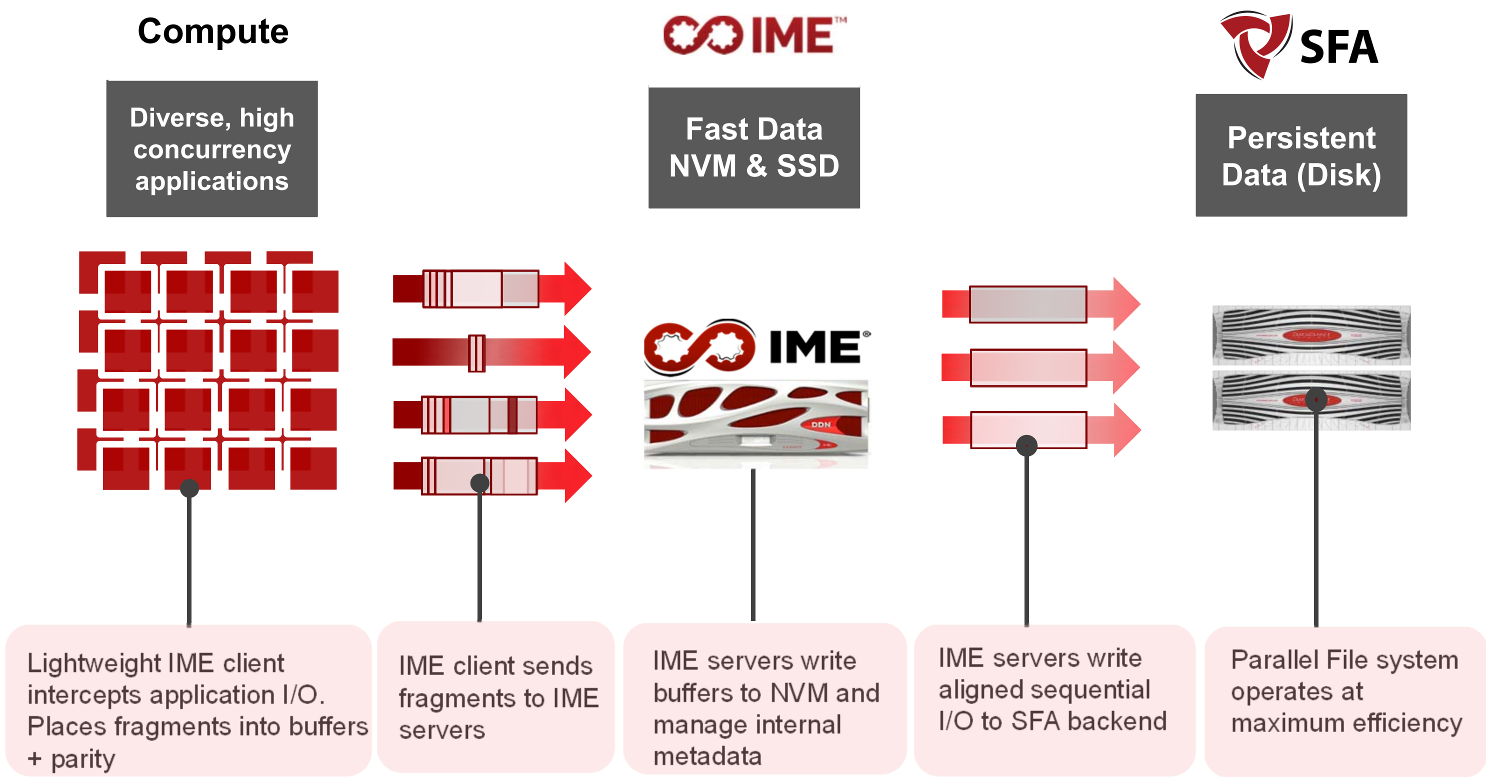
Parallel File System





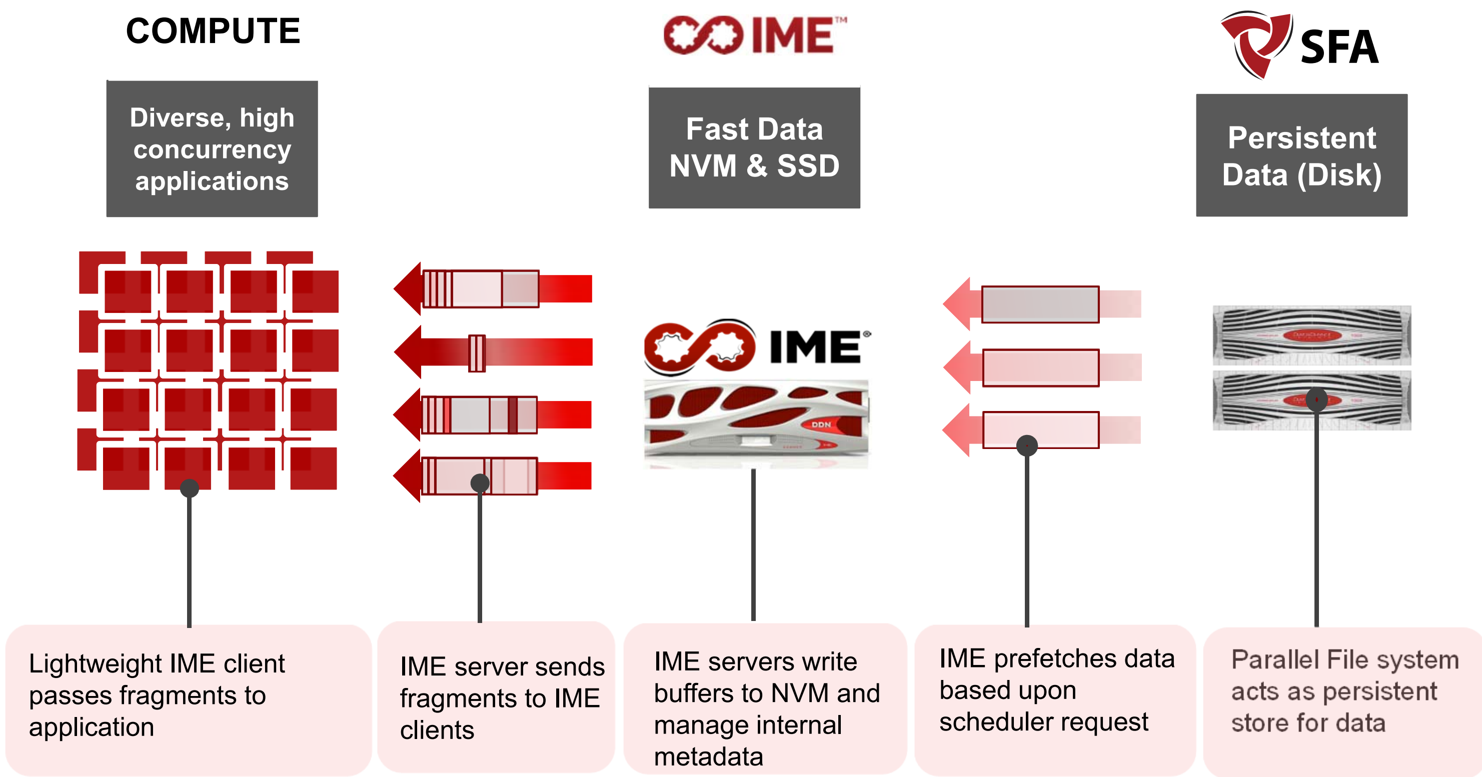
DDN | IME

Application I/O Workflow

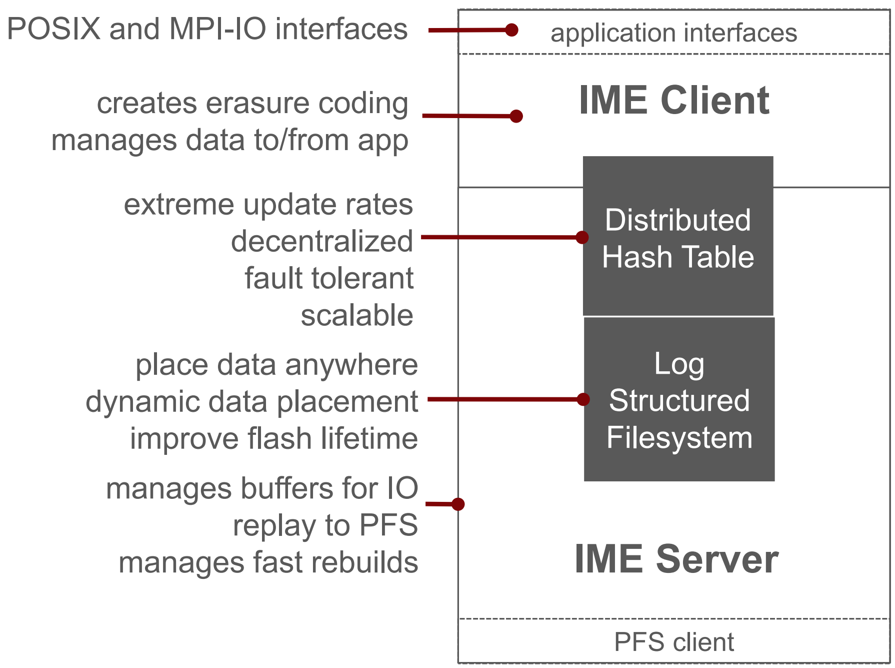


DDN | IME

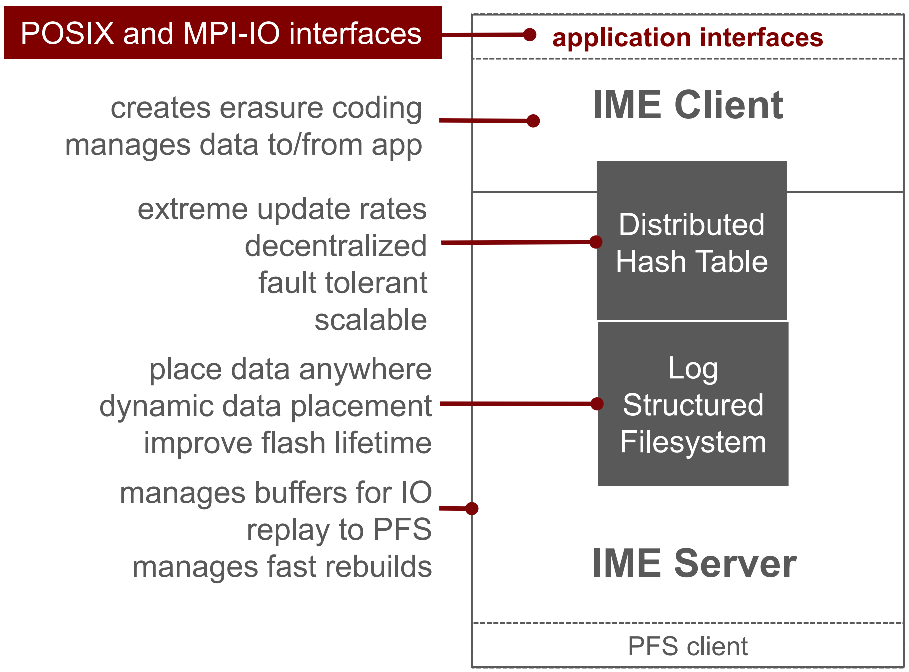
Application I/O Workflow



IME Architecture

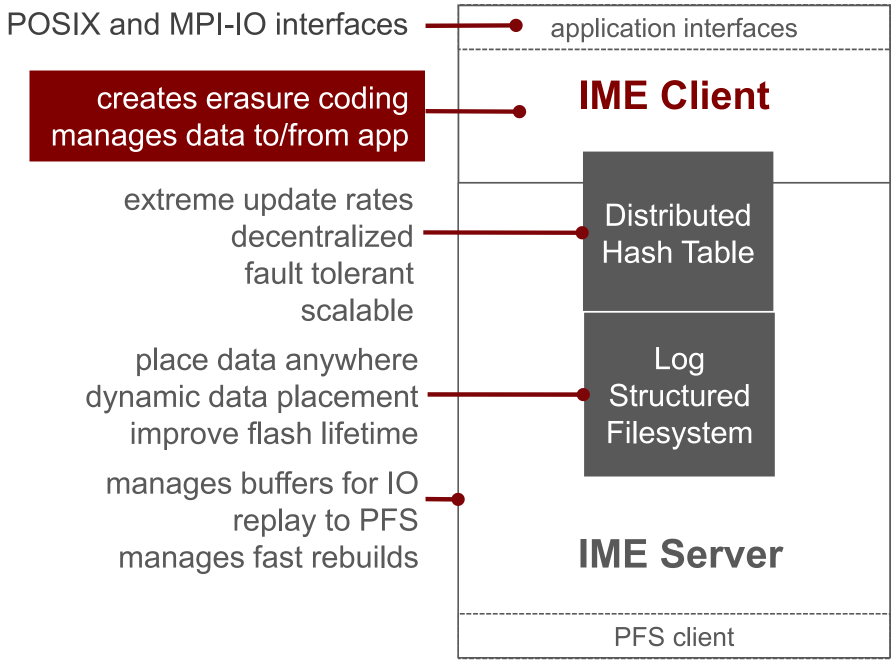


IME Architecture



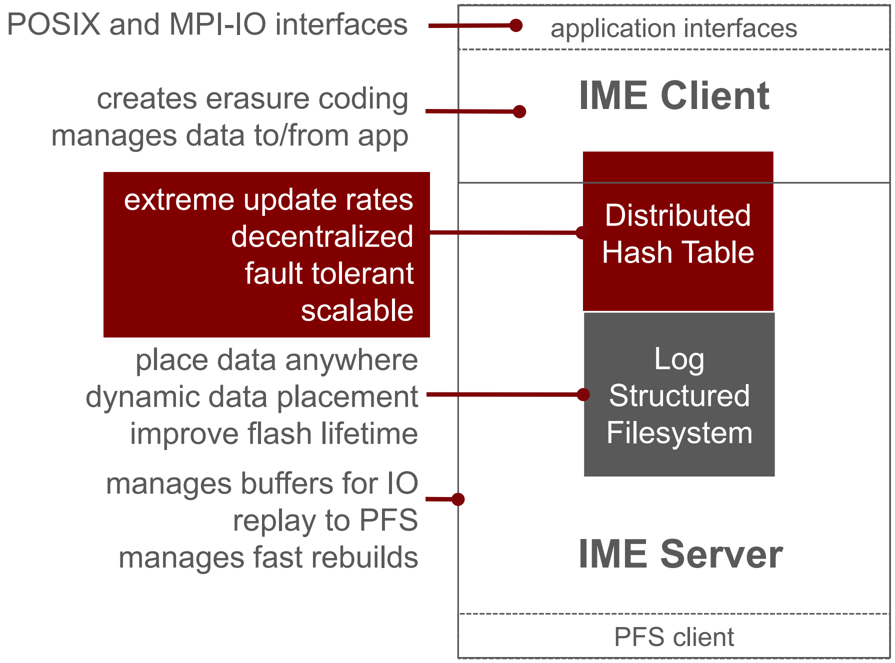
- No application changes required
- Applications must be recompiled/relinked to utilize IME

IME Architecture



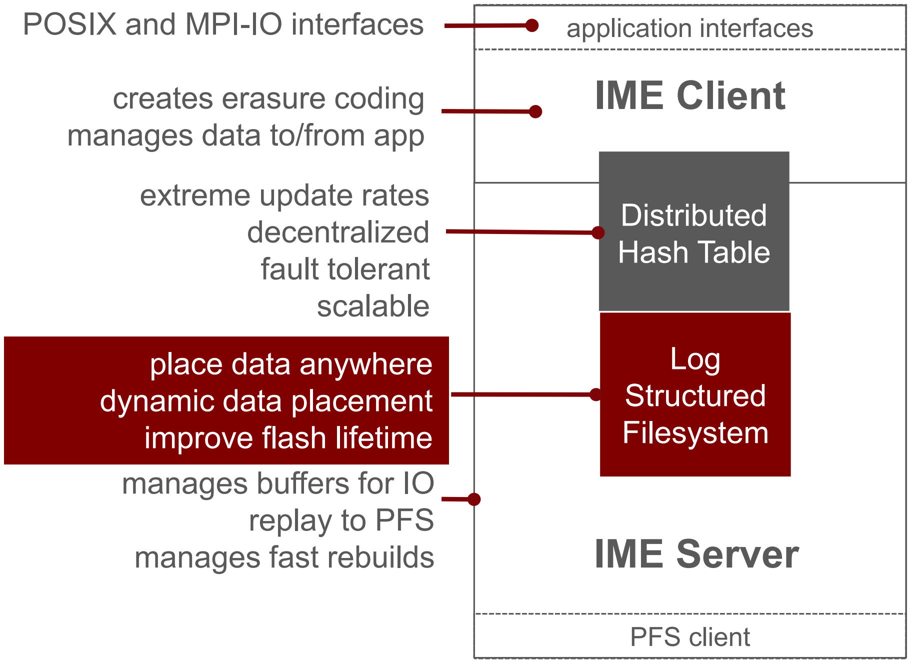
- **Client buffers IO fragments and sends to IME servers**
- **also (optionally) creates parity buffers according to RAID scheme chosen**
- **sends application and data to IME servers such that loss of a server or SSD does not result in data loss**
- **Heuristics in IME clients can intelligently pre-fetch IME-resident data to applications**

IME Architecture



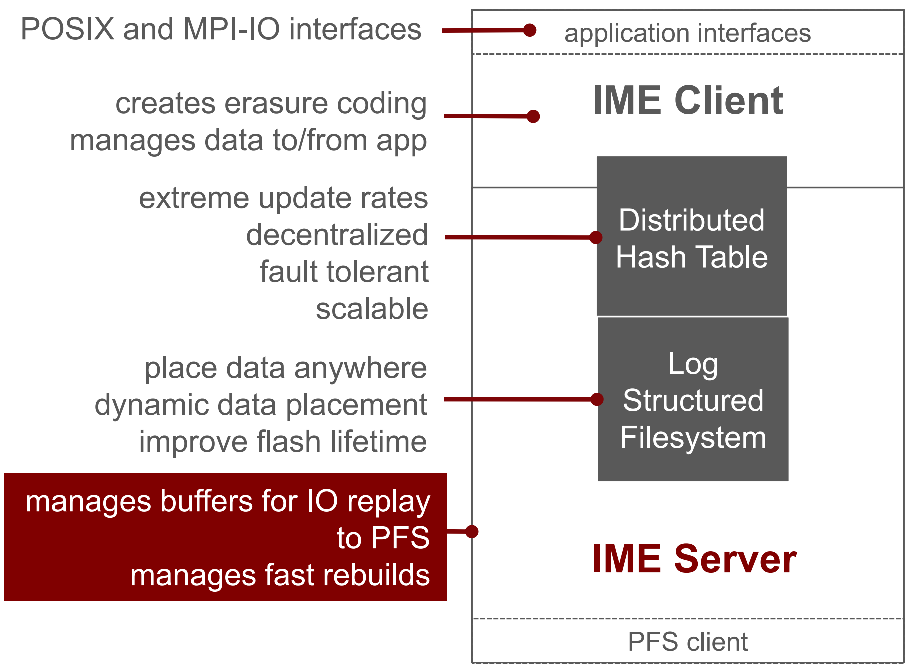
- **DHT is at the core of IME**
- **distributed index manages locations of files and objects**
- **Unique routing and data placement properties differentiate IME DHT from other DHTs**
- **Fast O(1) routing algorithm built on high-performance, non-cryptographic hash algorithms**
- **Load is uniformly distributed across DHT nodes**

IME Architecture



- **Data can be placed anywhere within IME – any server, any SSD**
- **IOPs to SSDs are minimized and optimized for NAND Flash**

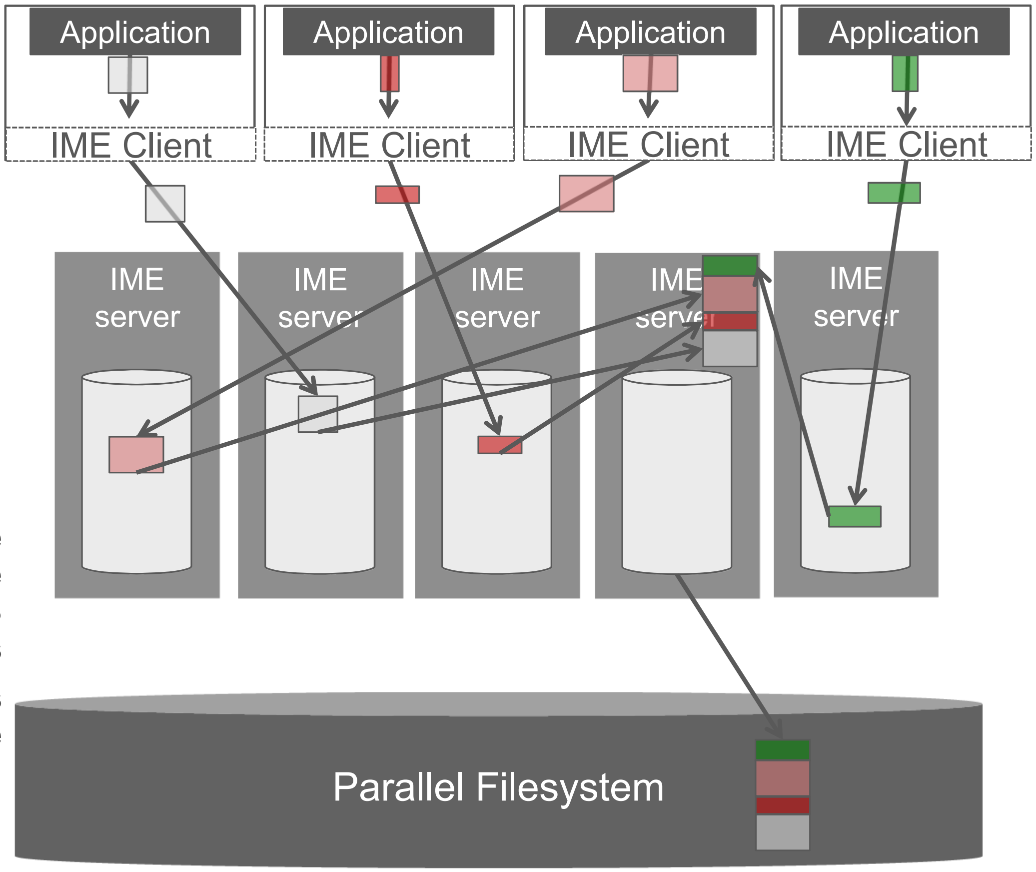
IME Architecture



- Actively reorganize I/Os to coalesce small block I/Os into large chunks and perform full stripe alignment before submission to the back end, Read acceleration is almost the reverse process
- Protect data by disseminating erasure coded chunks across the IME appliance cluster
- Pre-fetch data into IME from PFS using out-of-band commands and APIs

IME Dataflow

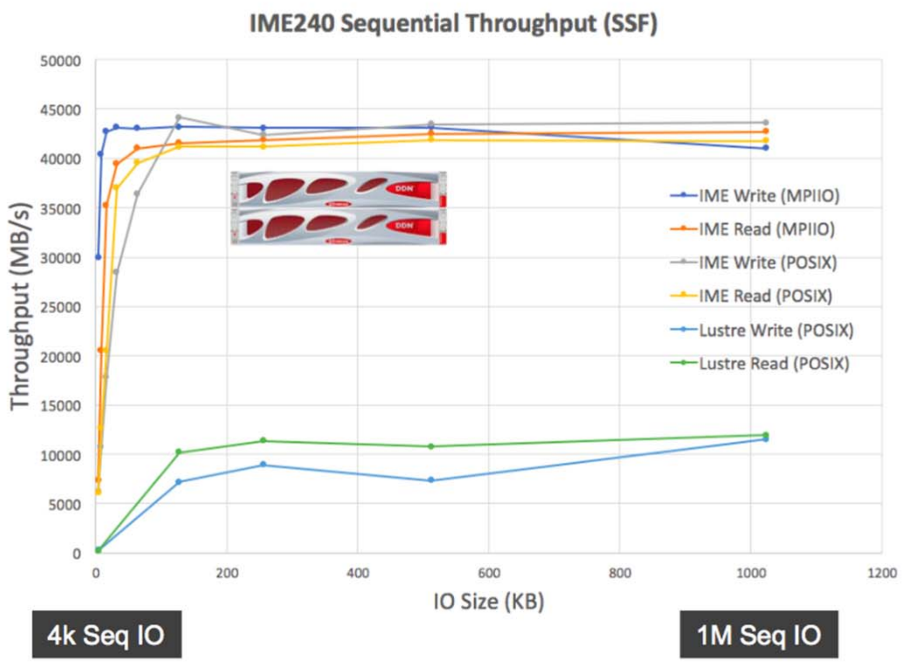
1. Application issues fragmented, misaligned IO
2. IME clients send **fragments** to IME servers
3. Fragments to IME servers accessible via DHT to all clients
4. Fragments to be flushed from IME are assembled into PFS stripes
5. PFS receives complete PFS stripe



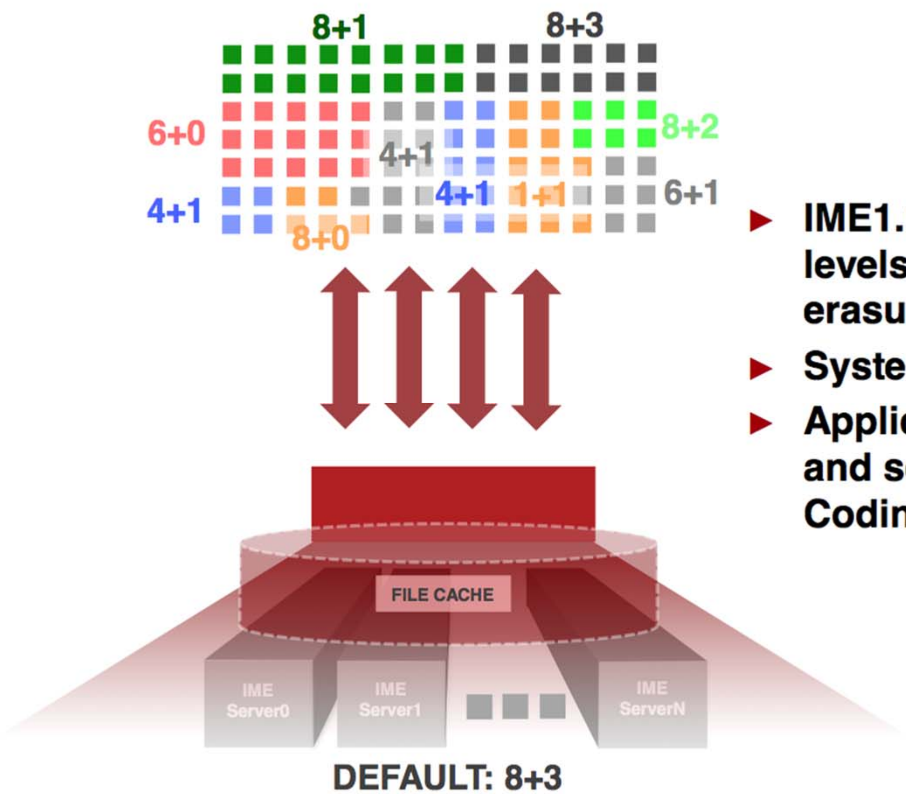
15

IME – Blistering I/O Performance

- 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**



IME – Erasure Encoding



- IME1.1 supports multiple resilience levels through flexible, adaptive erasure coding
- System Wide Default up to 15+3
- Applications can overide defaults and select a specific Erasure Coding Scheme

IME – System Options

- **IME 240**
  - 2U commodity server
  - Up to 23 NVMe SSDs per system
  - Up to 23GB/s, read and write
  - IB/OPA/Ethernet support
- **IME 140**
  - 1U commodity server
  - Up to 8 NVMe SSDs per system
  - Up to 20GB/s read, 11GB/s write
  - IB/OPA/Ethernet support
- **IME is a software product first and foremost.**
- **Linear scaling (assuming a capable network)**
  - Largest system is ~1PB capacity, 1.2TB/s throughput. 50 IME servers.

18

I/O Challenges: Locking on a parallel file system

I/O Challenges

Application Developer approach	IO Characterisation	IO Challenge
Multi-physics	Non-trivial mix of IO behaviours	Filesystem must be optimised for all IO types, not just a small subset
Workflows and Ensembles	Coordination of files/objects across many jobs. Different IO access patterns from different elements of the workflow	Requires a global namespace with strong consistency
Adaptive mesh refinement	Very difficult to manage and maintain logical application "block" sizes with underlying filesystem	Non-optimal access to files, incurs RMW, fs lock contention
Metadata-orientation	Small, random IOs, Shared file formats	Limited by filesystem locking
Machine learning	High read activity	Tough for HDD: disk thrashing
Higher Concurrency and heterogeneous CPUs	Large thread counts make sequential IO look like random IO	avoids all assistance from Filesystem and Storage caches

File Organization and Concurrency – Distributed Lock Managers

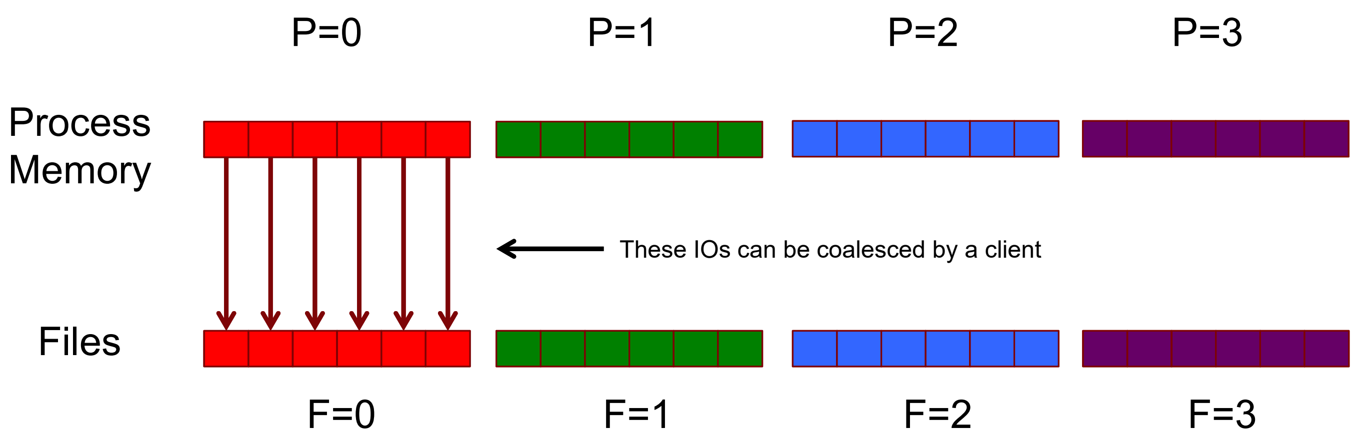
Mode	NL	CR	CW	PR	PW	EX
NL	Yes	Yes	Yes	Yes	Yes	Yes
CR	Yes	Yes	Yes	Yes	Yes	No
CW	Yes	Yes	Yes	No	No	No
PR	Yes	Yes	No	Yes	No	No
PW	Yes	Yes	No	No	No	No
EX	Yes	No	No	No	No	No

DEC's VMS Distributed Lock Manager Truth Table

- **Many PFS's use distributed lock managers to protect shared access to data**
- **Multiple readers are OK (use of "Protected Read" (PR))**
- **Single writer for atomic access (use of "Protected Write" (PW))**

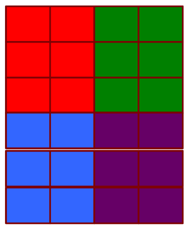
21

File per Process



Example: ior –b 4K –t 4K -F

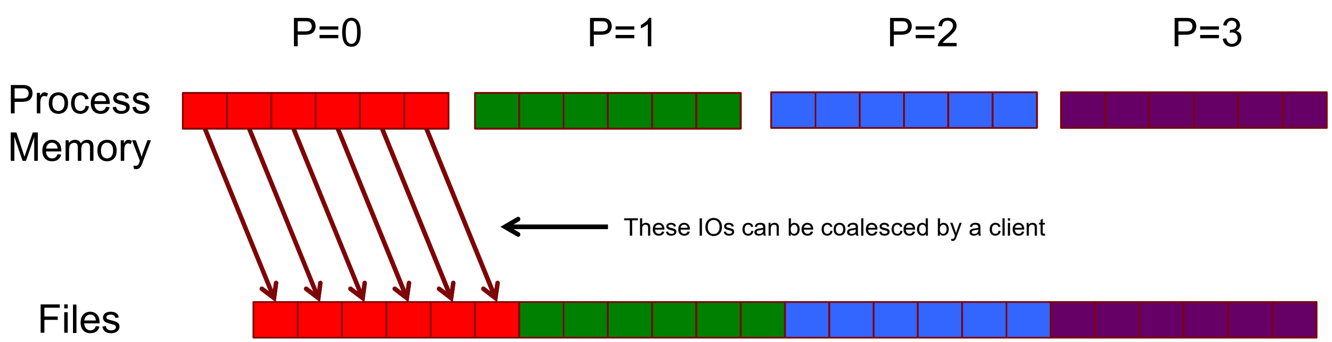
Tile-oriented Organization



File per Process – PFS

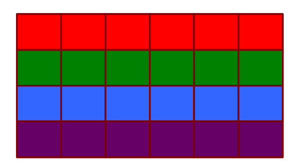
File per Process – IME

Segmented, Shared File

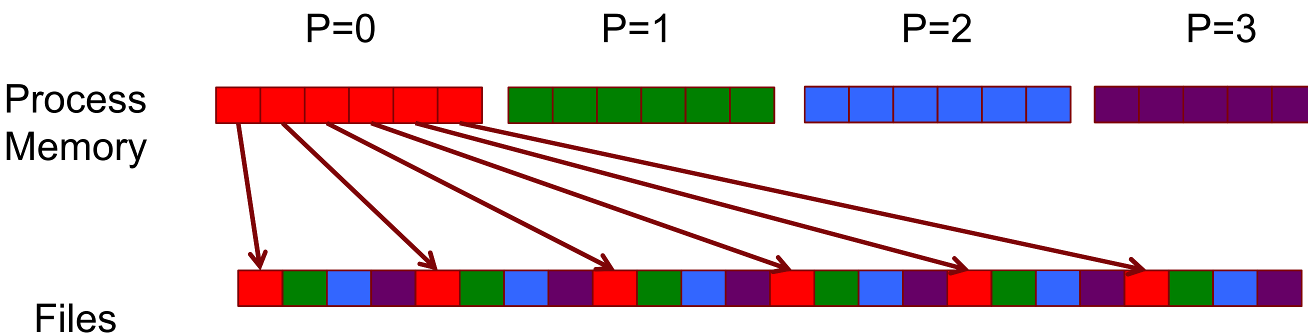


Example: ior –b 4K –t 4K

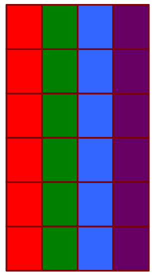
Row-oriented Organization



Strided / Interleaved, Shared File

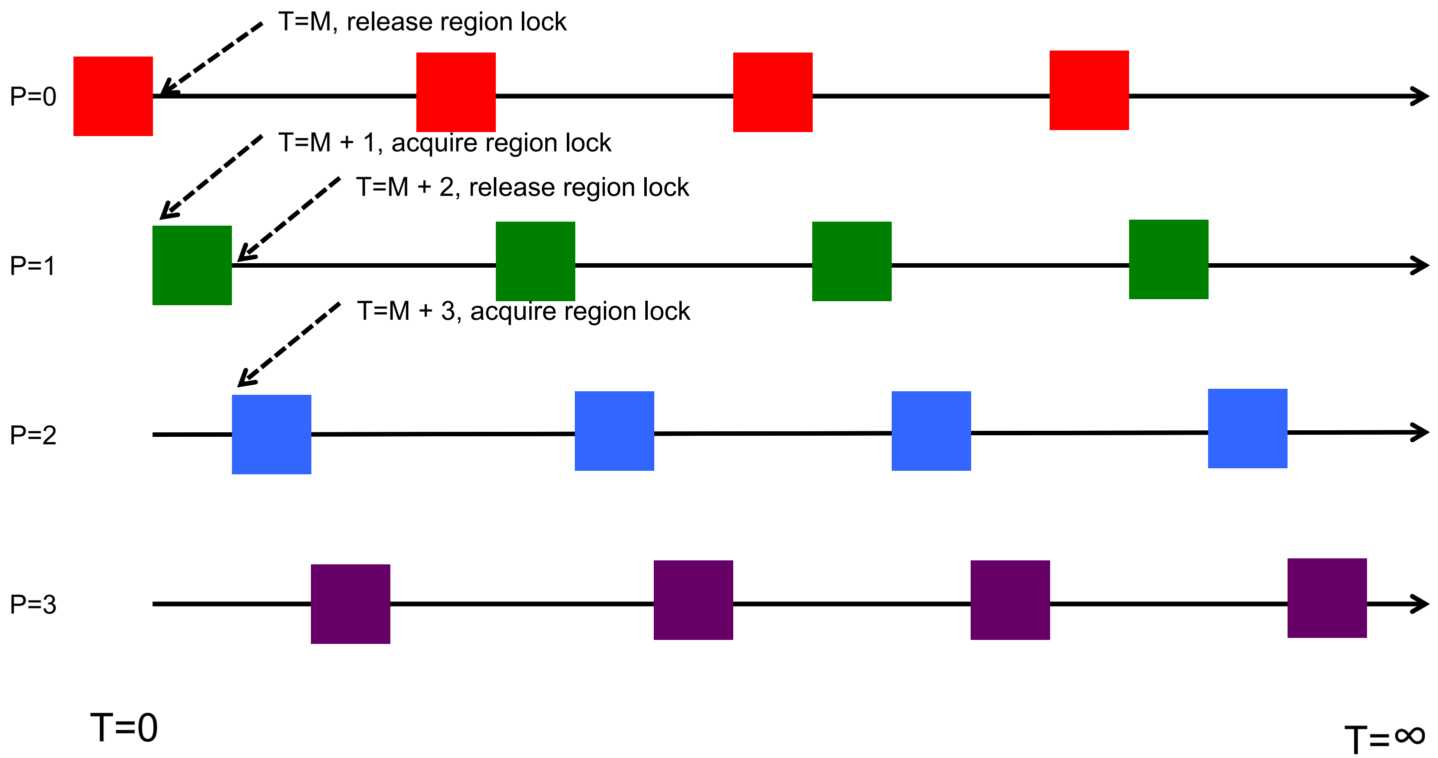


Column-oriented Organization



Example: ior –b 4K –t 4K –S

Strided / Interleaved, Shared File – PFS



Observations:

- Atomic access may induce false-sharing and cache flushes on remote processes
- This pattern is often referred to as “shuttling”

Strided / Interleaved, Shared File – IME

Observations:

- IME does not require locking as it (1) log-structures the IO requests, (2) builds a map of the logged IO and original location, and (3) transactions orders the arrival / consistency of the log as it arrives at IME servers
- Non-contiguous IOs are aggregated at the client and server to reduce transaction costs

Pseudo-random IO Patterns, Shared File – Adaptive Mesh Refinement and Irregular Mappings

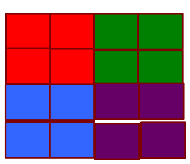
App Rank and IO Order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
App Rank to File Offset Map	12	8	13	9	4	0	5	1	14	10	15	11	6	2	7	3

B F B F B F B F B F ← Forward and Backward Seek Pattern

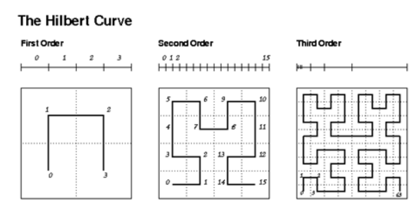
IO Order	12	4	14	6	8	0	10	2	13	5	15	7	9	1	11	3

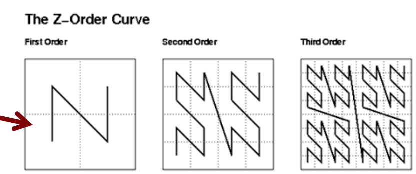
← Lock Contention

Tile-oriented

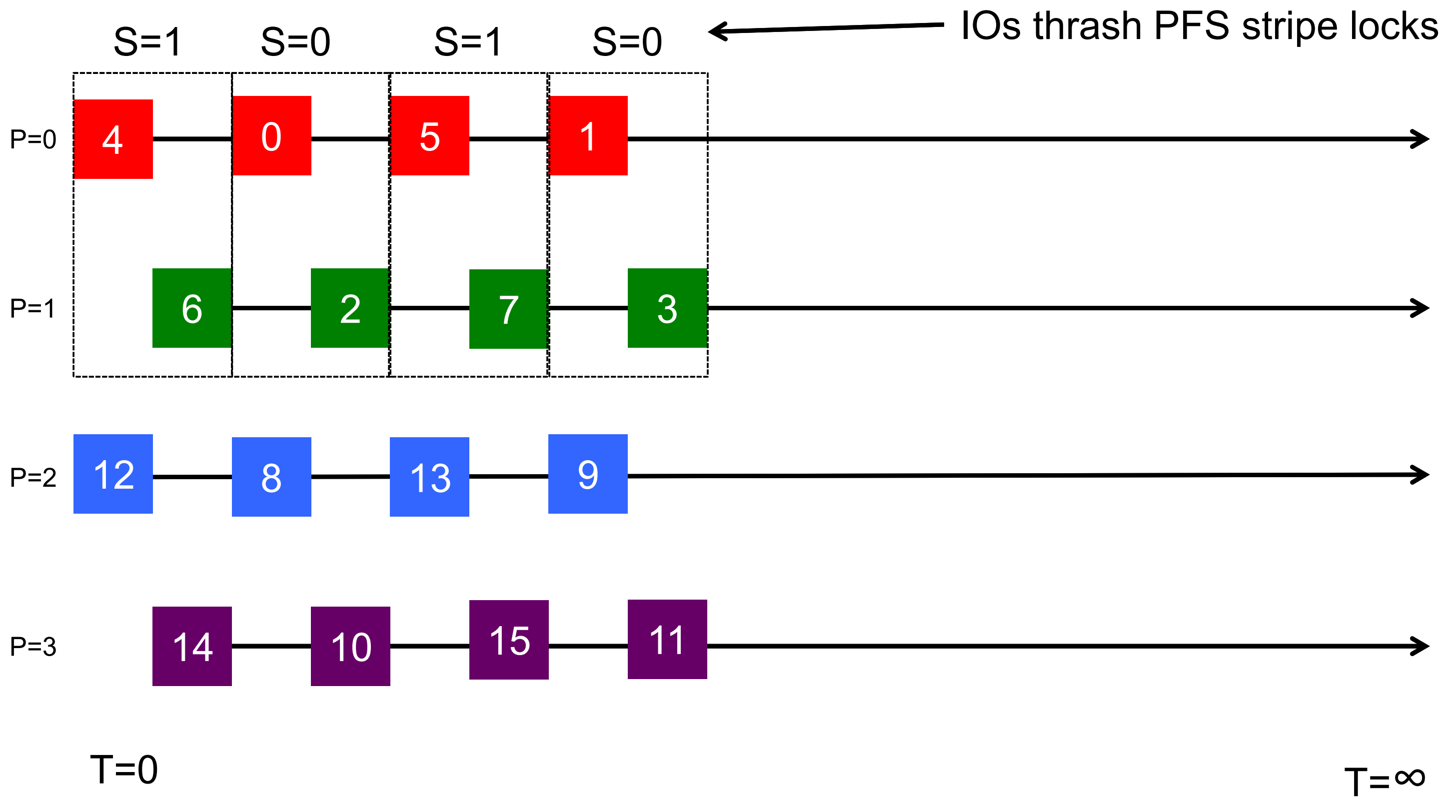


This example uses a Z-Order Curve to assign applications ranks





Pseudo-random IO Patterns, Shared File – PFS



Pseudo-random IO Patterns, Shared File – IME

Some small scale results

Note: experimental numbers only

IME vs All Flash Lustre, equipment

- **(4) IME 240s, EDR connected, (80) NVMe drives.**
- **(1) ES14KX, (2) OSSs, EDR connected, (20) all flash OSTs (RAID1).**
- **2 EDR connected clients**
  - Lab time was tight, and someone decided to play with the infiniband fabric yesterday.

- **Goal here is to show effects of page sized locking on Lustre (similar behavior on GPFS)**
- **Lustre client tunings:**
  - Max_pages_per_rpc=4096,max_rpcs_in_flight=32,checksum=0
  - Could've used better ratio of MPI ranks to OSTs for file-per-process

IME vs All Flash Lustre, baseline

- **IOR File per process, 1MB transfer sizes, file size 2x memory per node. Single node**

- **IME:**
  - Max Write: 7949.38 MiB/sec (8335.53 MB/sec)
  - Max Read: 11469.37 MiB/sec (12026.51 MB/sec)
- **Lustre all flash, stripe_count=1**
  - Max Write: 4281.58 MiB/sec (4489.56 MB/sec)
  - Max Read: 4300.66 MiB/sec (4509.57 MB/sec)

IME vs All Flash Lustre, baseline pt. 2

- **IOR File per process, 1MB transfer sizes, file size 2x memory per node. Two nodes**

- **IME:**
  - Max Write: 15796.31 MiB/sec (16563.63 MB/sec)
  - Max Read: 21455.67 MiB/sec (22497.90 MB/sec)
- **Lustre all flash, stripe_count=1**
  - Max Write: 4360.63 MiB/sec (4572.46 MB/sec)
  - Max Read: 7465.39 MiB/sec (7828.03 MB/sec)

IME vs All Flash Lustre, Shared File

- **IOR Shared file, 1MB transfer sizes, file size 2x memory per node.**

- **IME:**
  - Max Write: 7692.72 MiB/sec (8066.40 MB/sec)
  - Max Read: 11425.91 MiB/sec (11980.94 MB/sec)
- **Lustre all Flash, stripe_count=-1**
  - Max Write: 1249.74 MiB/sec (1310.44 MB/sec)
  - Max Read: 2640.12 MiB/sec (2768.36 MB/sec)

IME vs All Flash Lustre, Shared File Hard

- **IOR Shared file, 63560 byte transfer sizes, random, file size 2x memory per node.**
- **Why 63560? Per 1GB of data, start->end of each write (or read) overlaps with a multiple of 4096 only 32 times!**
- **Example command line (number of ranks handled by Slurm and PMI-2):**
  - /opt/ddn/ior/bin/IOR-mvapich -w -r -a POSIX -g -v -o /ssdfs/crusher/scratch/ior_ssdfs_random_shared_63560 -Q 1 -g -G 27 -k -e -s 100000 -b 63560 -t 63560 -v -E -e –z
- **IME:**
  - Max Write: 7692.72 MiB/sec (8066.40 MB/sec)
  - Max Read: 1759.89 MiB/sec (1845.38 MB/sec)
- **Lustre all Flash, stripe_count=-1**
  - Max Write: 906.33 MiB/sec (950.36 MB/sec)
  - Max Read: 1496.44 MiB/sec (1569.13 MB/sec)

IME vs All Flash Lustre, Shared File Hard pt. 2

- **IOR Shared file, 63560 byte transfer sizes, random, file size 2x memory per node. Two nodes**
- **IME:**
  - Max Write: 12416.78 MiB/sec (13019.93 MB/sec)
  - Max Read: 3511.12 MiB/sec (3681.68 MB/sec)
- **Lustre All Flash, stripe_count=-1**
  - Max Write: 263.18 MiB/sec (275.97 MB/sec)
  - Max Read: 172.68 MiB/sec (181.07 MB/sec)

Bonus, simultaneous shared file 50/50 writes/reads via homegrown app

- **iohitman – I wrote this for "fun" due to IOR not being flexible enough to demonstrate performance on less HPC-like workloads**
  - Python, mpi4py
  - Uses FUSE mountpoint (no IME Native kernel bypass)
  - 2 MPI communicators, one for writes, one for reads. 50/50 split. 160 total MPI ranks, 10 nodes
  - Shared file, random I/O, I/O sizes between 4KB and 1MB. **Unaligned,** unless the random integer happens to be divisible by 4096 bytes. Completely random range of I/O sizes
  - Ran in Sept, 2017
- **Results:**
  - INFO TEST_PARAMS: cmd_line: -b 134217728 -k --test-name 31 -o /tmp/imeadmin_fuse/crusher_scratch/iohitman_outfile.dat -i 4096,1048576 -s 1073741824 --tags date=1505549149,ioprofiler=0
  - **mixed_shared_random_READ**, ranks: 80, total_gb_rw: 81, **mbs: 8727.14827433**, **iops: 17508.1678343** test_time: 9.50413553119
  - **mixed_shared_random_WRITE**, ranks: 80, total_gb_rw: 81, **mbs: 15466.8313767**, **iops: 31029.1370211** test_time: 5.36270151138

39

IO-500 results

Some real scaling numbers

IO-500

- **Website:**
  - https://www.vi4io.org/std/io500/start
- **Source code:**
  - https://github.com/VI4IO/io-500-dev
- **First Annual List:**
  - https://www.vi4io.org/io500/start
- **Detailed results:**
  - https://www.vi4io.org/_media/events/2017/sc-bof-bent.pdf
- **2017 Winner:**
  - IME @ JCAHPC (Japan)! 25 IME14Ks (~50 IME240s), Omnipath fabric.
  - 2.5 racks of equipment
  - Original IOR benchmark yielded 1.2TB/s writes and reads

Breaking down IO-500 Results, Metadata

#	information				io500	mdtest							find
	system	institution	filesystem	client nodes	md	easy create	easy stat	easy delete	hard create	hard read	hard stat	hard delete	hard
					kIOP/s	kIOP/s	kIOP/s	kIOP/s	kIOP/s	kIOP/s	kIOP/s	kIOP/s	kIOP/s
1	Sonasad	IBM	Spectrum Scale	10	102.43	57.22	342.33	47.56	21.57	632.98	529.90	85.34	130.12
2	JURON	JSC	BeeGFS	8	89.81	193.37	718.18	150.61	8.42	0.00	100.85	8.76	302.99
3	Seislab	Fraunhofer	BeeGFS	24	68.55	103.15	433.14	172.95	5.38	13.87	57.40	13.87	215.02
4	Mistral	DKRZ	Lustre	100	46.64	18.15	153.05	7.74	17.80	37.58	156.07	8.80	912.86
5	Shaheen	Kaust	DataWarp	300	33.17	50.71	49.38	48.89	11.40	0.00	38.73	18.92	43.20
6	Shaheen	Kaust	Lustre	1000	31.03	12.66	120.81	14.96	13.67	0.00	127.32	11.30	61.62
7	Serrano	SNL	Spectrum Scale	16	27.98	32.55	303.02	26.15	2.29	0.00	25.20	26.15	34.47
8	EMSL Cascade	PNNL	Lustre	126	25.59	17.75	61.26	15.63	16.14	23.59	57.04	19.43	23.66
9	Oakforest-PACS	JCAHPC	IME	2048	19.04	28.29	54.20	35.88	1.51	57.38	61.50	0.95	186.69

** Courtesy of: https://www.vi4io.org/_media/events/2017/sc-bof-bent.pdf*

Breaking down IO-500 Results, Throughput

#	information				io500	ior			
	system	institution	filesystem	client nodes	bw	easy write	easy read	hard write	hard read
					GiB/s	GiB/s	GiB/s	GiB/s	GiB/s
1	Oakforest-PACS	JCAHPC	IME	2048	471.25	742.38	427.41	600.28	258.93
2	Shaheen	Kaust	DataWarp	300	151.53	969.45	894.76	15.55	39.09
3	Shaheen	Kaust	Lustre	1000	54.17	333.03	220.62	1.44	81.38
4	Mistral	DKRZ	Lustre	100	22.77	158.19	163.62	1.53	6.79
5	JURON	JSC	BeeGFS	8	14.24	30.42	48.36	1.46	19.16
6	Seislab	Fraunhofer	BeeGFS	24	5.13	18.79	22.34	0.89	1.86
7	EMSL Cascade	PNNL	Lustre	126	4.88	17.81	30.19	0.39	2.72
8	Sonasad	IBM	Spectrum Scale	10	4.57	34.13	32.25	0.17	2.33
9	Serrano	SNL	Spectrum Scale	16	0.65	1.08	1.03	0.22	0.71

** Courtesy of: https://www.vi4io.org/_media/events/2017/sc-bof-bent.pdf*

Breaking down IO-500 Results, Rankings

#	information				io500		
	system	institution	filesystem	client nodes	score	bw	md
					sqrt(GiB*kIOP)/s	GiB/s	kIOP/s
1	Oakforest-PACS	JCAHPC	IME	2048	101.48	471.25	19.04
2	Shaheen	Kaust	DataWarp	300	70.90	151.53	33.17
3	Shaheen	Kaust	Lustre	1000	41.00	54.17	31.03
4	JURON	JSC	BeeGFS	8	35.77	14.24	89.81
5	Mistral	DKRZ	Lustre	100	32.15	22.77	46.64
6	Sonasad	IBM	Spectrum Scale	10	21.63	4.57	102.43
7	Seislab	Fraunhofer	BeeGFS	24	18.75	5.13	68.55
8	EMSL Cascade	PNNL	Lustre	126	11.17	4.88	25.59
9	Serrano	SNL	Spectrum Scale	16	4.25	0.65	27.98

** Courtesy of: https://www.vi4io.org/_media/events/2017/sc-bof-bent.pdf*

Takeaways

- **Try this at home, and take a look at the Lustre Distributed Locking Manager (LDLM) processes on a client node**
- **All Flash can perform quite well, but not for all workloads**
- **Maybe Lustre Lock Ahead can help with this:**
  - Lock ahead - Request extent locks from userspace
    - Credit: Cray
  - Landed Lustre 2.11
  - https://jira.hpdd.intel.com/browse/LU-6179
- **IME was designed to solve these parallel file system challenges, as well as provide predictable performance at high levels of concurrency. Disk based systems are subject to seek penalties as concurrency increases, even with well-formed sequential workloads.**
- **Adding SSDs to file systems can help, but file systems seem to be the bottleneck between achieving good application facing random I/O performance and concurrency**

45

Questions?