Whamcloud

Overstriping:
Extracting Maximum Shared File Performance

Patrick Farrell, LUG ‘19

DDN® STORAGE

Scaling I/O Performance

Whamcloud

- Some basics:
  Lustre data performance is scaled by adding OSTs (and metadata performance by adding MDTs)
- I/O must be spread across OSTs to benefit
- Lustre can do this by using many files (file per process) or a single file, striped across many OSTs (single shared file)
- Either approach gets you access to many OSTs at the same time

Lustre File Striping

Whamcloud

- Lustre allows striping of file data across multiple disk targets (OSTs)
- Horizontal scaling of I/O performance within a file, not only for multiple files
- RAID0 striping across OSTs, one stripe per OST
- Originally limited to 160 stripes, now allows up to 2000 stripes per file
- Can put all OSTs in one file, so can get full performance… right?

Single Shared File vs File Per Process (FPP)

Whamcloud

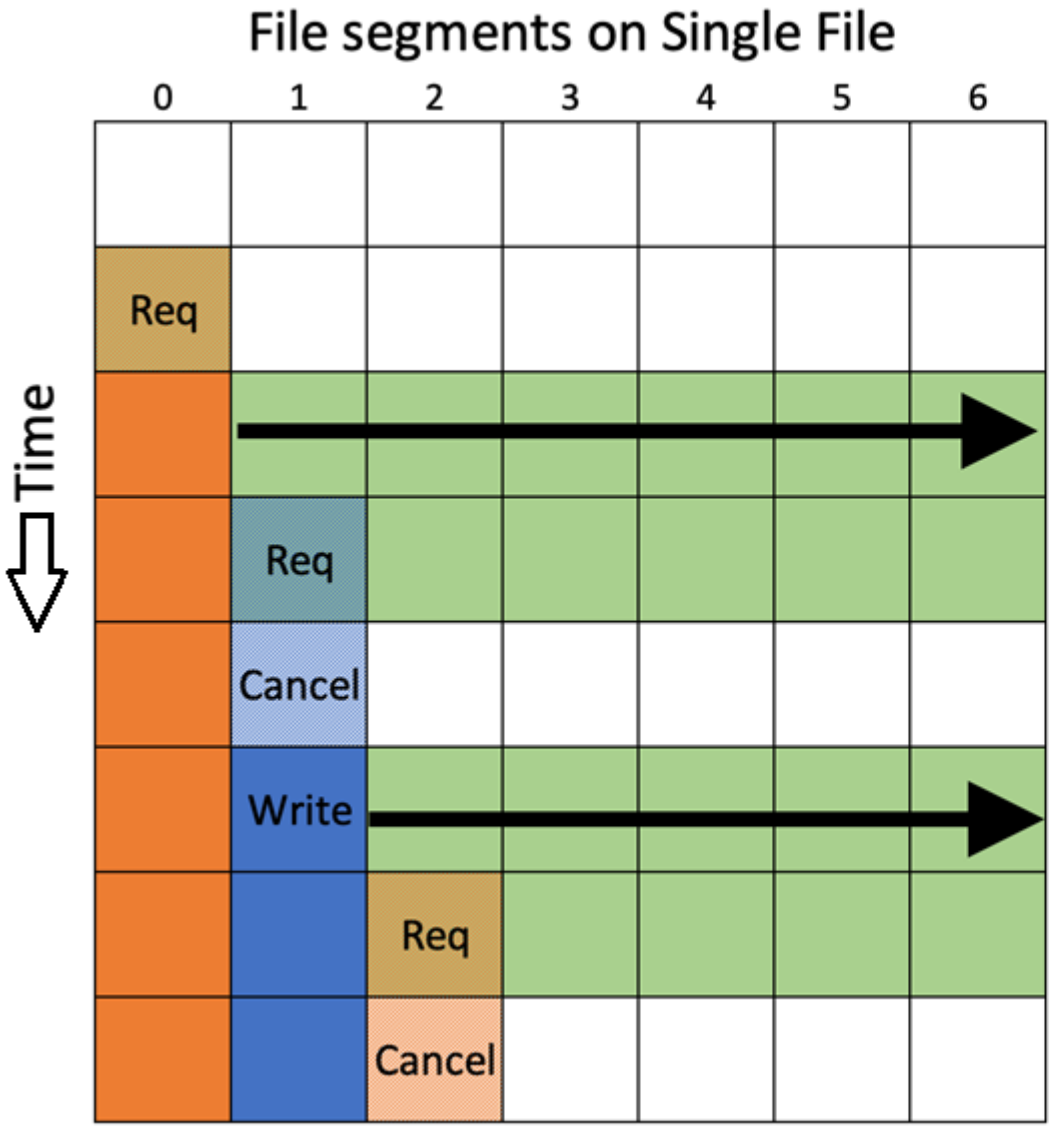
- File per process gives a fully independent I/O domain for each process
- All writing can happen without lock contention with other clients
- Single shared file means many writers to the same file
- Each stripe has its own locking, and Lustre supports range locking…
- Note:
  Because read locks can overlap, shared file read performance doesn't have this issue. Unless specified, we're talking about writing.

Shared File Writing

Whamcloud

- ‘Good’ shared file I/O generally means strided I/O (ex., MPIIO/MPICH library collective buffering)
- Writes are non-overlapping, clients write alternating blocks in a strided pattern
- In practice, it doesn’t scale at > 1 writer per stripe
- Best bandwidth achieved at 1 writer, with I/O aligned to stripes
- Writers otherwise end up doing “lock exchange”
- Can only scale by adding more stripes

Shared File Locking – Two client example



Two clients doing strided writes to same file

Client 1 request to write to segment 0

No locks current on file
Server expands lock by cllent 1 to whole flle

Client 2 requests to write to segment 1

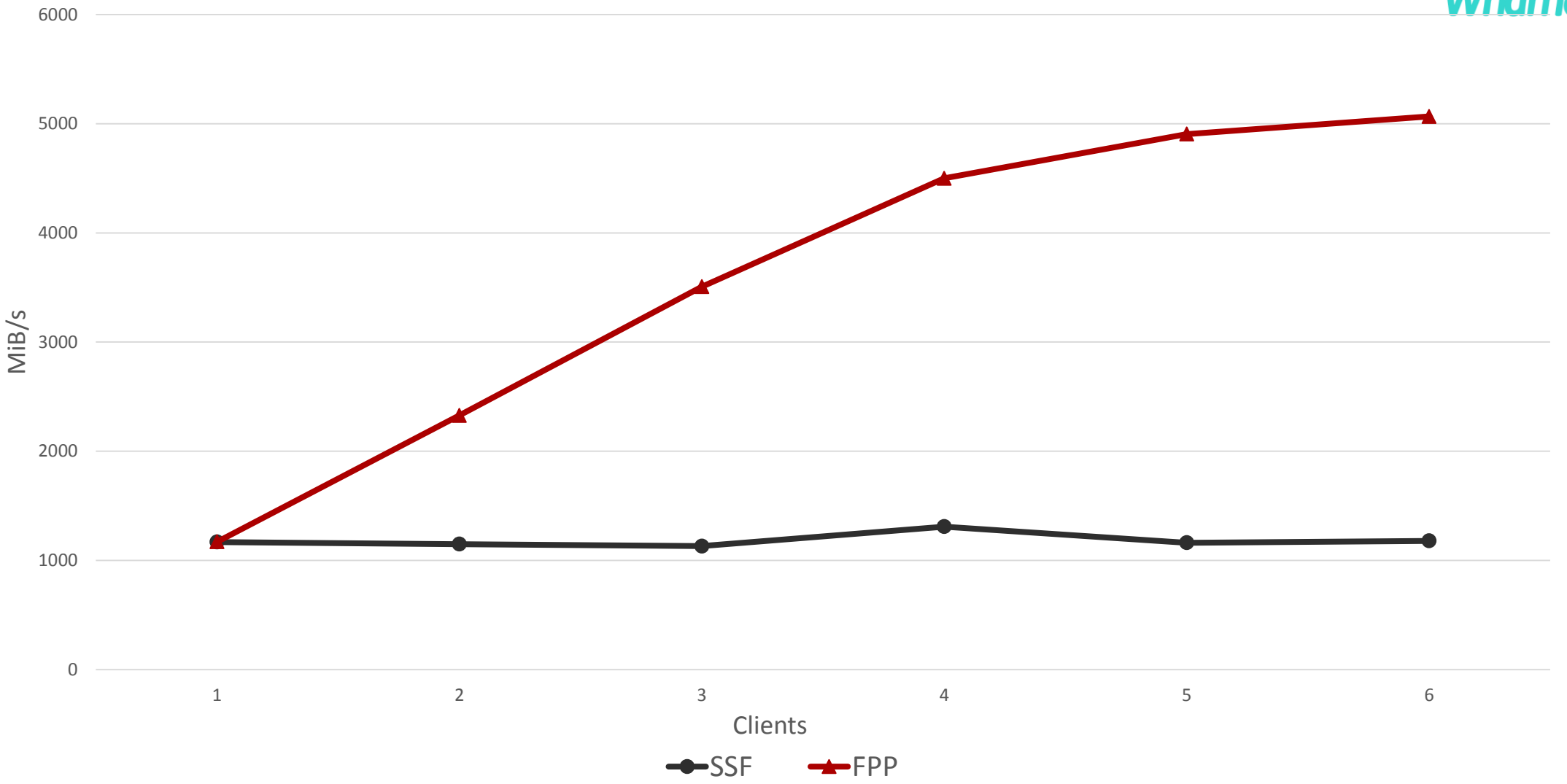
Lock assigned to client 1 is revoked
Client 2 lock request is processed

No locks current on file
As before, server expands lock by cllent 2 to whole flle

Client 1 request to strided write to segment 2
Lock assigned to client 2 revoked

Extent lock contention repeats throughout I/O
I/O is completely serialised with no parallel strided writes

SSF Write Scaling: Single OST, SSF vs FPP



Extracting Maximum OST Performance

Whamcloud

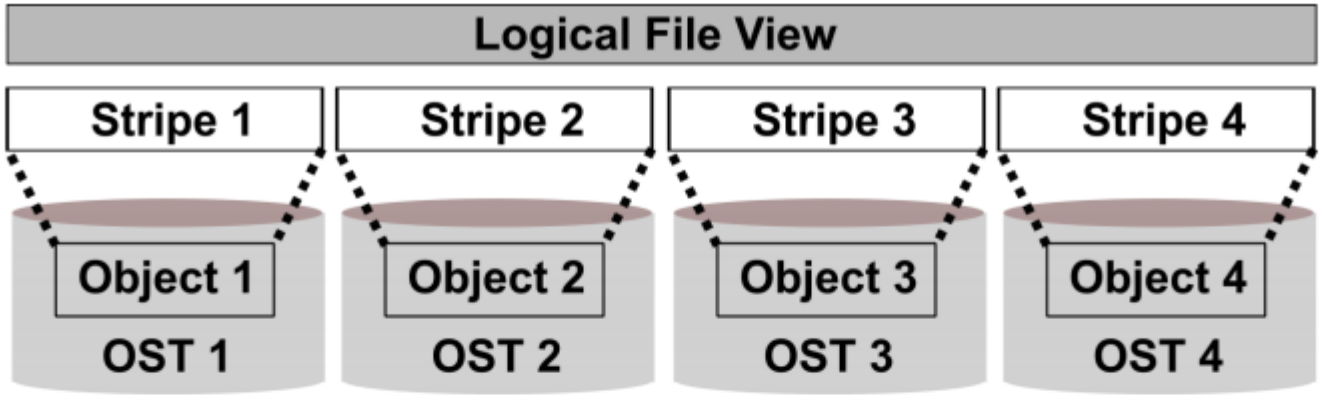
- OSTs today are 5-10 GiB/s write, next gen 10-30 GiB/s
- Distributed parity enables huge OSTs (512 TB+)
- Getting maximum performance means many files per OST (FPP)
- With 1 writer per OST, SSF is stuck way behind
- We use many stripes per OST in the FPP config
- Just one in the SSF config...

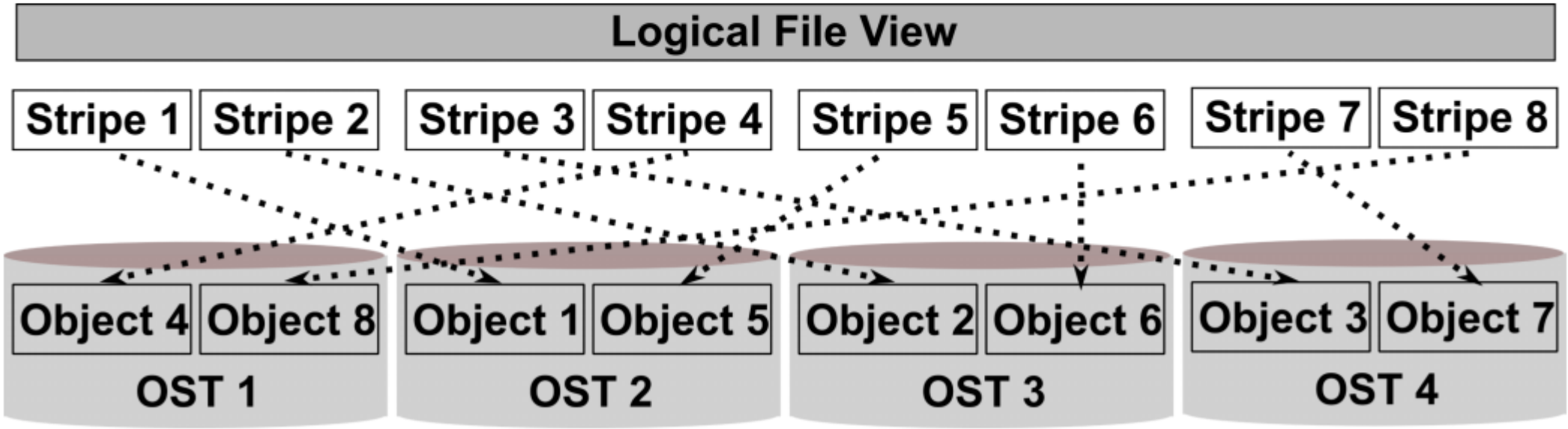
Overstriping: Stripe != OST

Whamcloud

- No reason why we must have only one stripe per OST
- Reasons are all historical
  - Contention on HDD based OSTs with multiple files/stripes
  - Unnecessary for slow OSTs (1 stripe is plenty)
  - Inertia from copying traditional disk level RAID0
- Overstriping means num stripes > num OSTs, ie, > 1 stripe per OST
- Basic change is trivial:
  Remove explicit checks **preventing** this
- Revealed several latent bugs with high stripe counts and xattr handling, but no architectural changes required

Overstriping: Graphically





Usage

Whamcloud

- Overstriping is easy – Uses existing commands and interfaces (setstripe, getstripe)
- It's just stripe count, with the option to overstripe if stripe count > OST count
- Like any other layout option:
  - Set using `lfs setstripe`
  - Works in `setstripe ioctl & llapi`
  - Works in default layouts (set on directories)

Examples: `lfs setstripe`

Whamcloud

- 600 stripes in a file – Use ‘C’ to request overstriping:

```
lfs setstripe -C 600 testfile
```

- Works with OST pools
  32 stripes, 4 OSTs in pool (8 stripes per OST):

```
lfs setstripe -C 32 -p 4_ost_pool testfile
```

- Can specify OSTs – 4 stripes on OST 2, 4 on OST 3:

```
lfs setstripe -o 2,3,2,3,2,3,2,3 testfile
```

Examples: `lfs getstripe`

- Getstripe – 2 OSTs (0 & 1), 4 stripes:

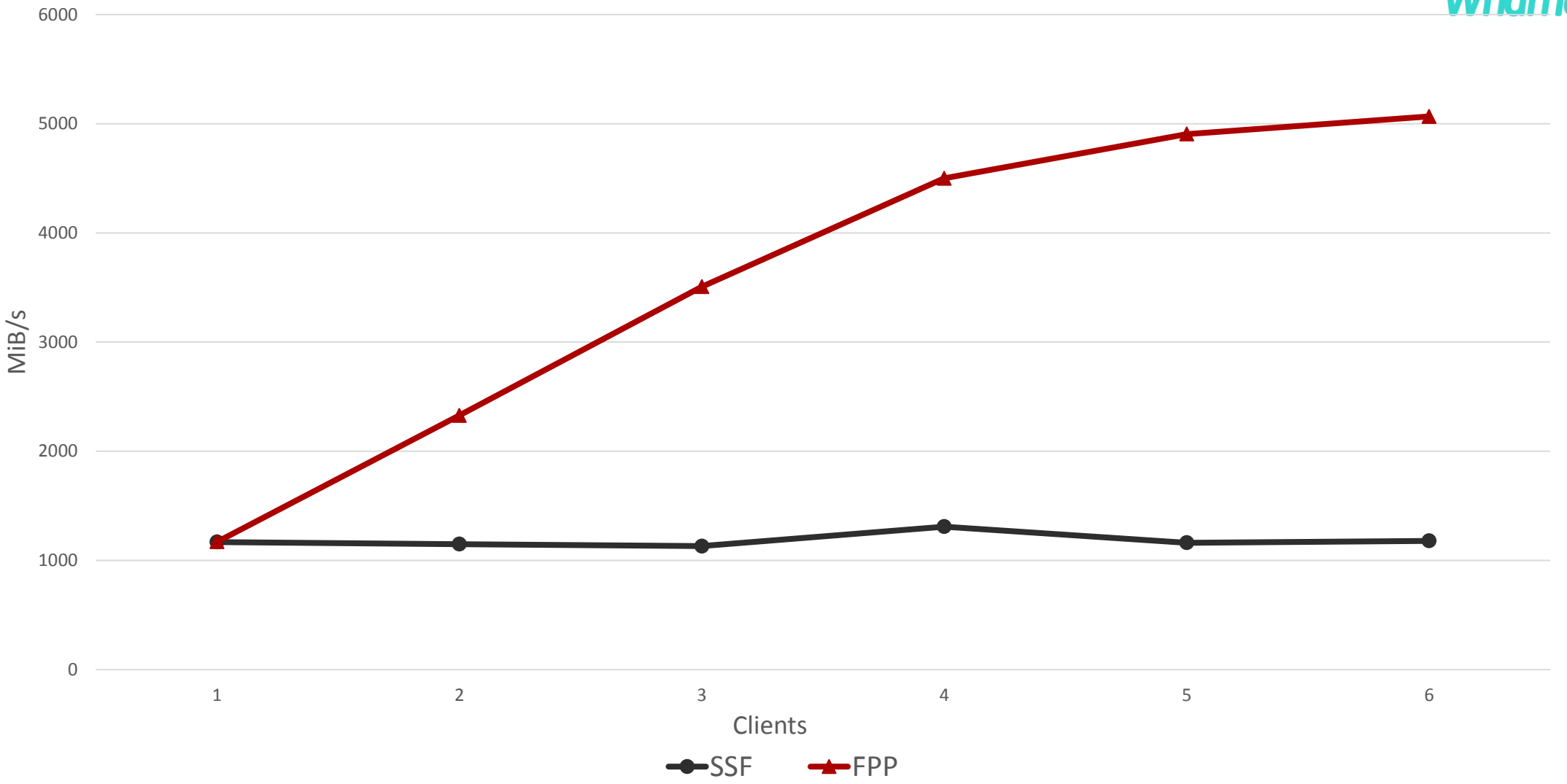
```
lmm_stripe_count:  4
lmm_stripe_size:   1048576
lmm_pattern:       raid0,overstripe
lmm_objects:
      - l_ost_idx: 0
        l_fid:     0x100000000:0x828:0x0
      - l_ost_idx: 1
        l_fid:     0x100010000:0x807:0x0
      - l_ost_idx: 0
        l_fid:     0x100000000:0x829:0x0
      - l_ost_idx: 1
        l_fid:     0x100010000:0x808:0x0
```

Benchmark Hardware

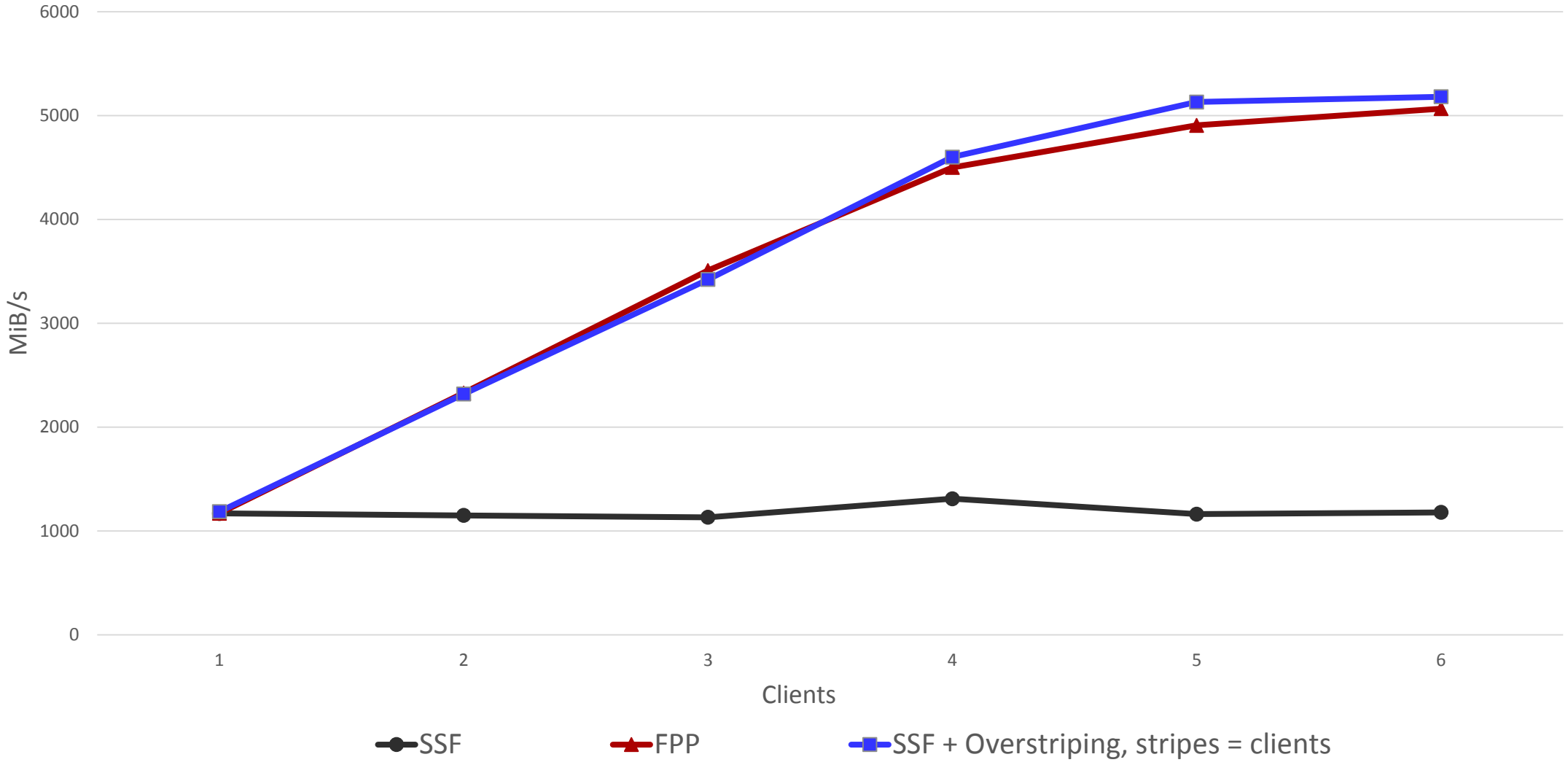
Whamcloud

- 1 x ES18K(SFA18KE)
  - OPA
  - 8 x SS9012 Disk enclosure
  - 640 x HGST 10TB NL-SAS(HUH721010AL4200)
- 4 x OSS (on Virtual Machine) with dual-rail on OPA
  - 5 x OST per OSS
- 2 x Lustre MDS
  - OPA
  - 1 x Intel Xeon Platinum 8160
  - 96GB DDR4 2667Mhz

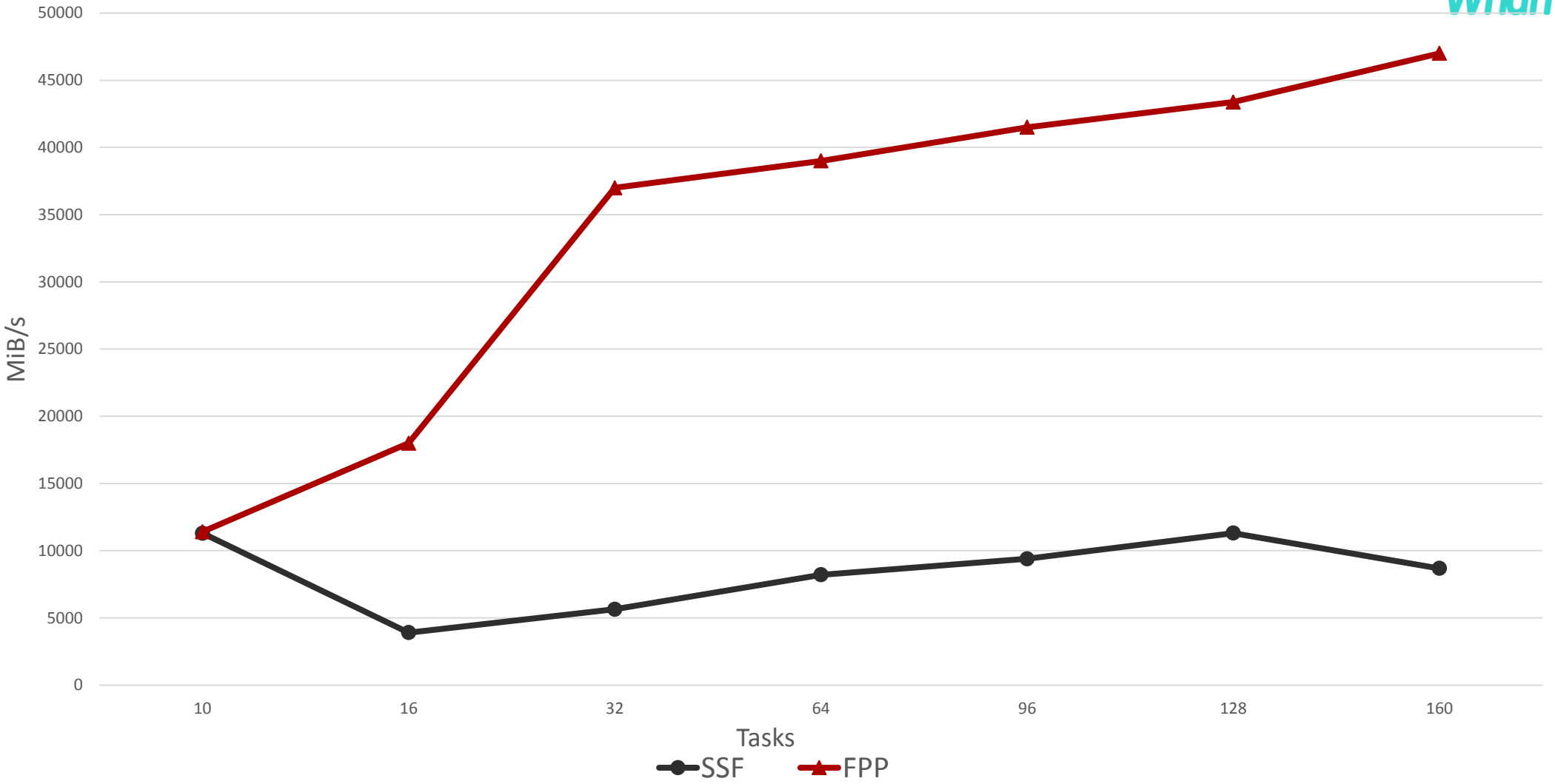
Benchmarking: Single OST, FPP vs SSF



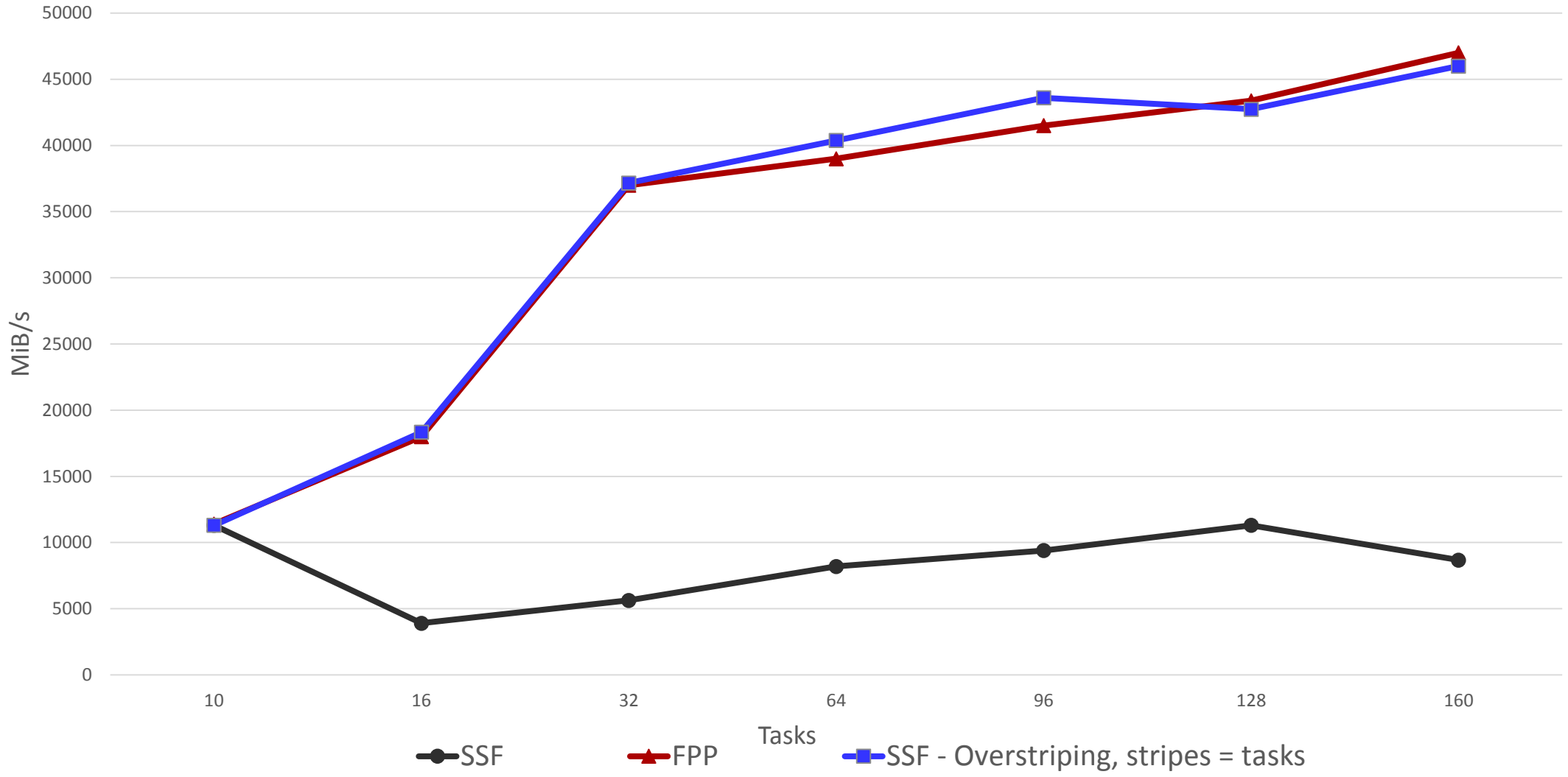
Benchmarking: Single OST, FFP vs SSF vs SSF + Overstriping



Benchmarking: 10 OSTs, FPP vs SSF



Benchmarking: 10 OSTs, FPP vs SSF vs SSF + Overstriping



Usage Recap

Whamcloud

- Primary usage:
  Extracting full file system performance in a SSF
  Only relevant for stripe count > OST count
  Must use stripe aligned writes
  - MPIIO collective buffering is helpful
- Becomes more important with faster OSTs
- Can be useful for small pools of very fast OSTs
  - For example, dynamically allocated per job pools

What about Lustre lock ahead?

Whamcloud

- Special Lustre locking feature introduced in 2.11 (LU-6179)
  - Uses manual lock requests to avoid ‘lock exchange’
  - Allows > 1 writer per stripe
- Very effective, but tricky to use
  - Requires MPIIO + Special library options
- Overstriping is simpler and covers most uses
- Lock ahead still relevant for extremely large systems
  - If you have 1000 OSTs, you can’t put 6 stripes per OST (2000 stripe limit)
- Combined with overstriping (stripe count++, writers per stripe++)

Limitations: Layout size

Whamcloud

- Adding stripes to a file increases the layout size
- Shared file means full layout is sent to all clients
- Compare FPP to SSF:
  - FPP: Total layout data to clients =
    1 stripe/file * 1 file/client * N clients = N*1 stripes
  - SSF: Total layout data to clients =
    N stripes/file * N clients = N^2 stripes
- Issue exists with widely striped files today, but only affects largest sites

Limitations: Layout size

Whamcloud

- Not nearly as bad as it sounds, most layouts are still pretty small
- Max layout size is 64 KiB, ~2700 stripes
- 160 stripes is ~ 4 KiB
- At moderate stripe counts, layout is so small it’s “free”, carried with open op without noticeable degradation

Potential Future Work: Layout size improvements

Whamcloud

- Lustre limited to 2000 stripes, because of XATTR size
- Layout is an xattr, 64 KiB limit per XATTR
- 2000 stripes is probably not enough for Exascale systems
- Solutions:
  - Simple: Add a second layout XATTR
  - Better: Compress layout. ~80% reduction in layout size. Fairly easy.
  - Best (But, high effort): Compact layouts (Derive OST FIDs from MDT FID)
- Compressed & compact layouts both reduce layout size, helps with open() problem

Potential Future Work, 2: Metadata Overstriping (LU-12273)

Whamcloud

- DNE 2 allows metadata striping
- If we have metadata striping, we can have metadata overstriping
- Allows greater performance within a single directory by placing > 1 stripe per MDT
- Considering for Lustre 2.14

Questions?

- Contact
  - Patrick Farrell, `pfarrell@whamcloud.com`
- Thanks to:
  Shuichi Ihara (DDN) for benchmark assistance
  Michael Moore (Cray) for CUG paper collaboration