xyratex

Advancing Digital Storage Innovation

Network Request Scheduler (NRS)

LAD ’12, Paris, France

Nikitas Angelinas
nikitas_angelinas@xyratex.com

Agenda

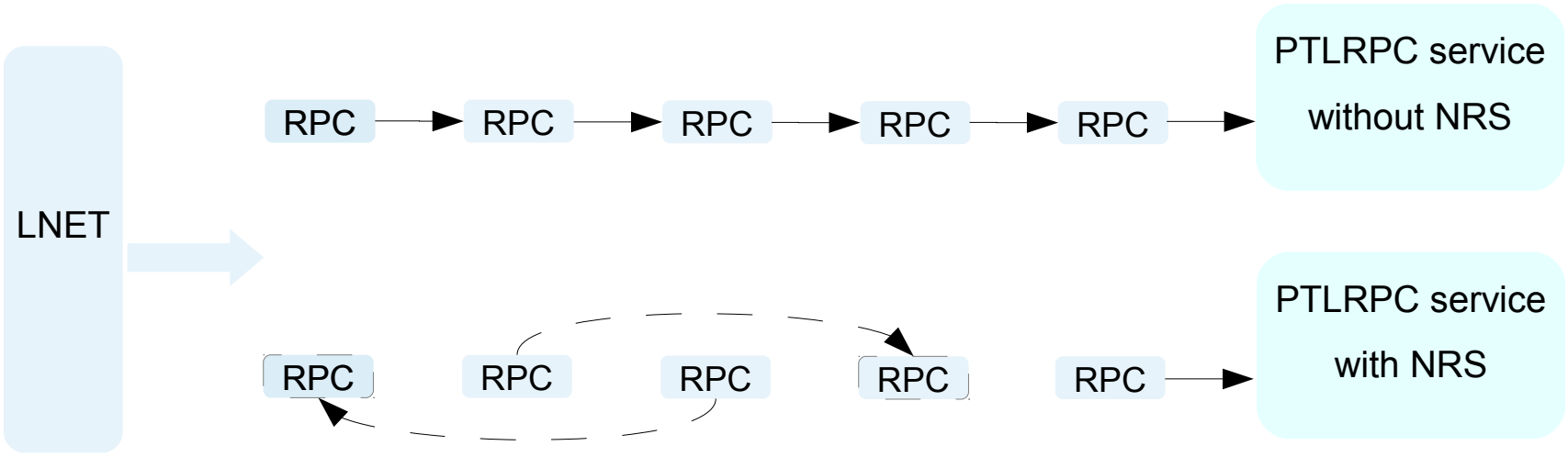
- NRS background
- SMP scaling considerations for NRS
- SMP scaling test results
- Misc
- Future tasks

Foreword

- NRS is a collaborative project between Intel and Xyratex
  - Code is at Intel's Gerrit server
  - Jira ticket LU-398
  - Targeting Lustre version 2.4
  - No regressions for functionality equivalent with current, non-NRS code
  - Testing indicates some beneficial use cases
  - Needs further large-scale testing and reviewing
  - Need to decide which SMP scaling (LU-56) NRS adaptation to merge

Concept

- NRS allows the PTLRPC layer to reorder the servicing of incoming RPCs
  - We are mostly interested in bulk I/O RPCs for now
- Solely server-based for now



Motivation

- Increased fairness amongst filesystem nodes, and better utilization of resources
  - Avoid starvation of clients
  - Load-balance RPCs across OSTs
  - Better network utilization
- Increased read throughput across the filesystem
  - Order brw RPCs according to their logical or physical offsets
  - Allows to reduce the amount of disk seeks in some cases
- Future applications; potentially based on future NRS framework revisions
  - Vary # of RPCs in flight depending on # of clients doing I/O
  - Other possibilities

NRS policies

- A binary heap data type is added to libcfs
  - Used to implement prioritized queues of RPCs at servers
  - Sorts large numbers of RPCs (10,000,000+) with minimal insertion/removal overhead (<2 usec)
- FIFO - Logical wrapper around existing PTLRPC functionality
  - Is the default policy for all RPC types
- CRR-E - Client Round Robin, RR over exports
- CRR-N - Client Round Robin, RR over NIDs
- ORR - Object Round Robin, RR over backend-fs objects, request ordering on logical or physical offsets
- TRR - Target Round Robin, RR over OSTs, request ordering on logical or physical offsets
- Other policies may be useful

ORR/TRR policies

- ORR serves bulk I/O RPCs in a Round Robin manner over available backend-fs objects
  - RPCs are placed in per-object groups of 'RR quantum' size; lprocfs tunable
  - Sorted within each group by logical of physical disk offset
  - ldiskfs physical offsets are calculated using extent information obtained via fiemap calls
  - Support for OST_READ and/or OST_WRITE RPCs; lprocfs tunable
- TRR is equivalent, but schedules RPCs in a Round Robin manner over available OSTs
- The main aim is to minimize disk seek operations, thus increasing read performance
- TRR may help with load balancing across OSTs
- ORR/TRR may take advantage of temporal and spatial locality

Framework features

- Allows to select a different policy for each PTLRPC service
  - Separate policy on HP and normal requests
- Policies can be hot-swapped via lprocfs, while the system is handling I/O
- Policies can fail in handling a request
  - Intentionally or unintentionally
  - A failed request is handled by the fallback, FIFO policy
  - FIFO cannot fail the processing of an RPC

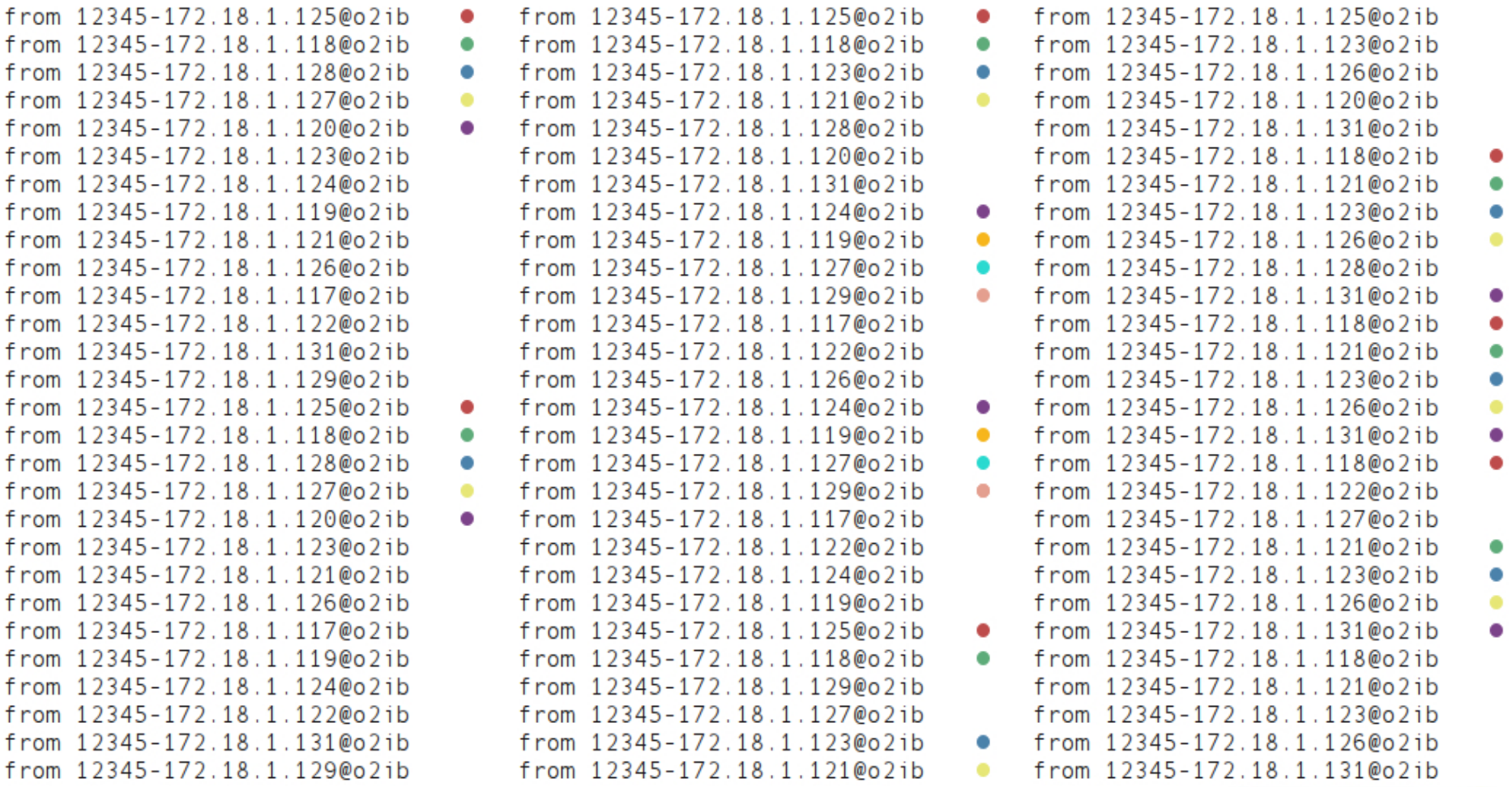
SMP scaling considerations

- MDS with CPTs performs much better; likely OSS with CPTs shows an improvement
  - NRS needs to be able to work well in multi-CPT servers
- Central scheduling entity (NRS) vs partitioned RPC handling (SMP scaling)
- Possible NRS implementations
  - Scheduler per CPT
  - Scheduler per service (one for all CPTs)
- Concerns
  - First option will inhibit the ability of NRS policies to enforce request ordering
  - Second option may defeat some of the benefits of having CPTs
    - In some cases may cause underutilization of some CPTs
    - Take CPT # into account during scheduling (not implemented yet)?
- Need to test each NRS policy in a variety of CPT configurations

TRR policy RPC distribution (TRR quantum = 8), NRS per CPT

CPTs = 1	CPTs = 2	CPTs = 4
from OST with index 3	from OST with index 2	from OST with index 3
from OST with index 3	from OST with index 2	from OST with index 0
from OST with index 0	from OST with index 2	from OST with index 0
from OST with index 0	from OST with index 3	from OST with index 0
from OST with index 0	from OST with index 2	from OST with index 0
from OST with index 0	from OST with index 3	from OST with index 0
from OST with index 0	from OST with index 2	from OST with index 3
from OST with index 0	from OST with index 3	from OST with index 1
from OST with index 0	from OST with index 3	from OST with index 3
from OST with index 0	from OST with index 3	from OST with index 1
from OST with index 3	from OST with index 3	from OST with index 3
from OST with index 3	from OST with index 3	from OST with index 3
from OST with index 3	from OST with index 1	from OST with index 1
from OST with index 3	from OST with index 1	from OST with index 3
from OST with index 3	from OST with index 1	from OST with index 2
from OST with index 3	from OST with index 1	from OST with index 2
from OST with index 3	from OST with index 1	from OST with index 2
from OST with index 3	from OST with index 2	from OST with index 2
from OST with index 2	from OST with index 2	from OST with index 3
from OST with index 2	from OST with index 1	from OST with index 3
from OST with index 2	from OST with index 1	from OST with index 2
from OST with index 2	from OST with index 1	from OST with index 2
from OST with index 2	from OST with index 1	from OST with index 2
from OST with index 2	from OST with index 1	from OST with index 3
from OST with index 2	from OST with index 1	from OST with index 3
from OST with index 2	from OST with index 1	from OST with index 3
from OST with index 1	from OST with index 0	from OST with index 3
from OST with index 1	from OST with index 1	from OST with index 3

CRR-N policy RPC distribution (14 clients), NRS per CPT

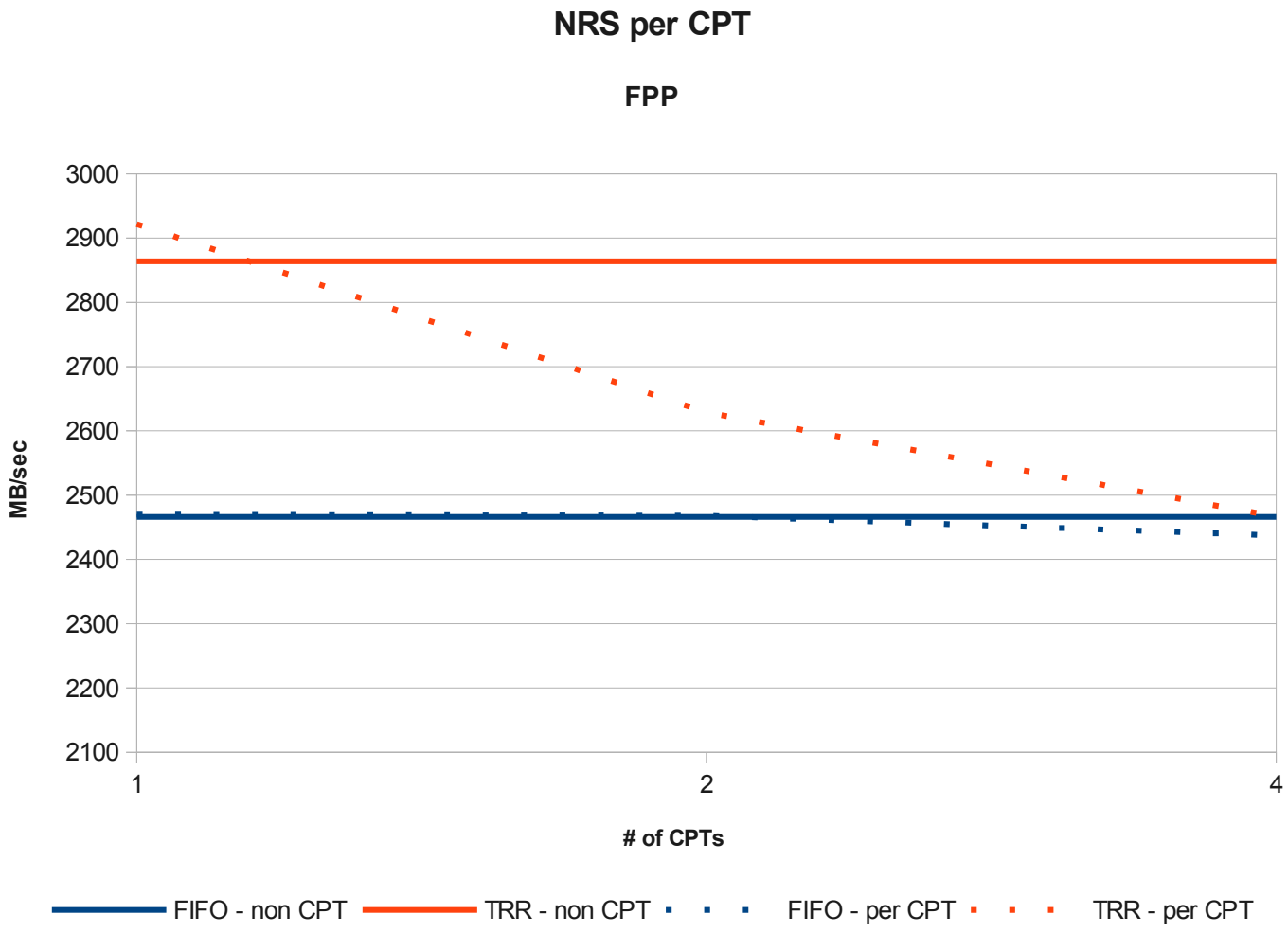


xyratex

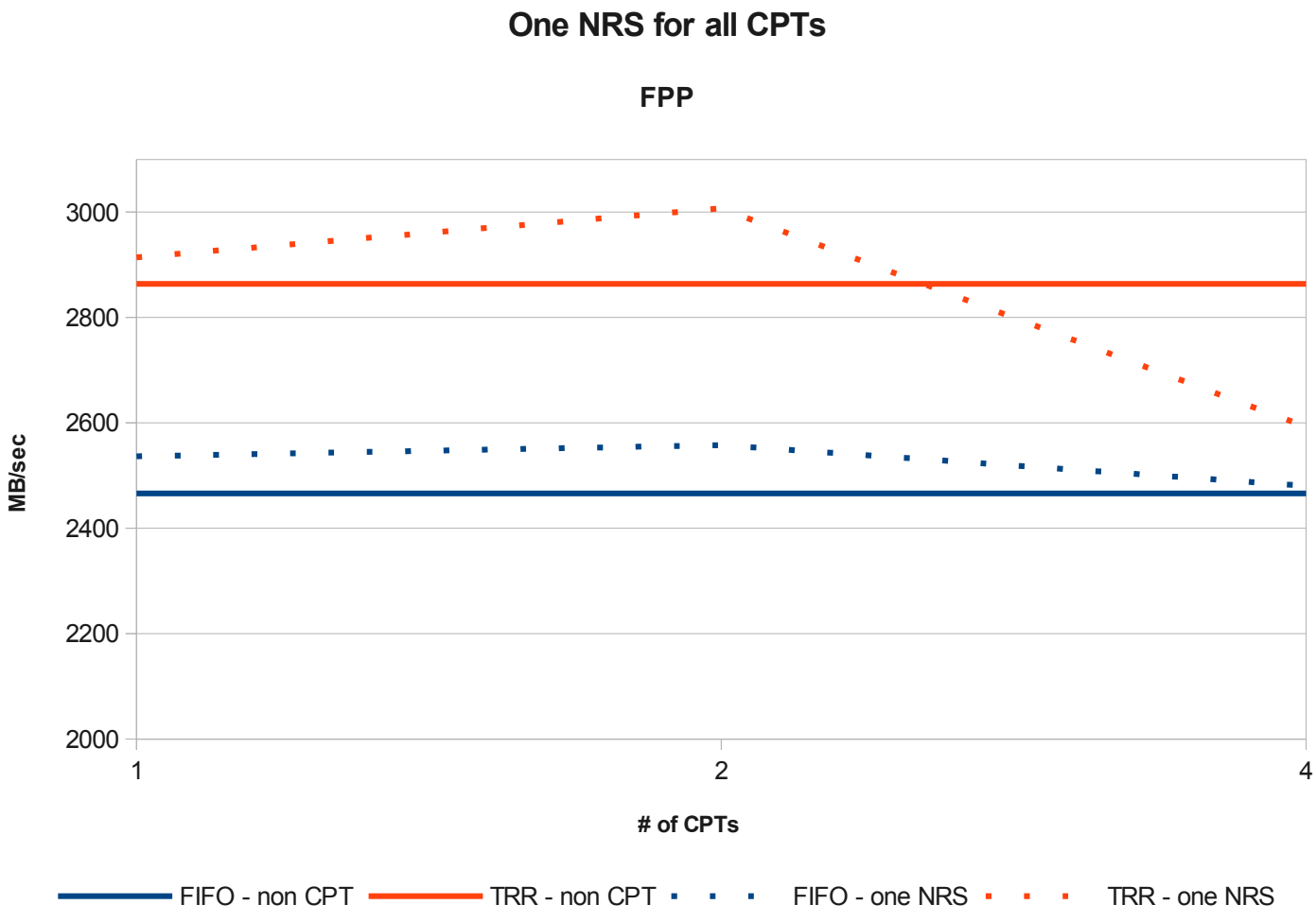
TRR policy tests

- CPT table defined using 'cpu_npartitions' libcfs parameter; using all available CPUs
- Performance is compared between FIFO and TRR for different CPT configurations
  - Using physical offset ordering, TRR quantum = 256
- IOR read test; each IOR process reads 16 GB of data in 1MB transfers
  - FPP and SSF, stripped directories
  - Client kernel caches cleared between reads
- 1 Xyratex CS3000, single-SSU (2 OSSs)
- Only 14 clients, read operations generate few RPCs
  - Using ost_io.threads_max=128 on both OSS nodes
- The OSS nodes are not totally saturated with this configuration

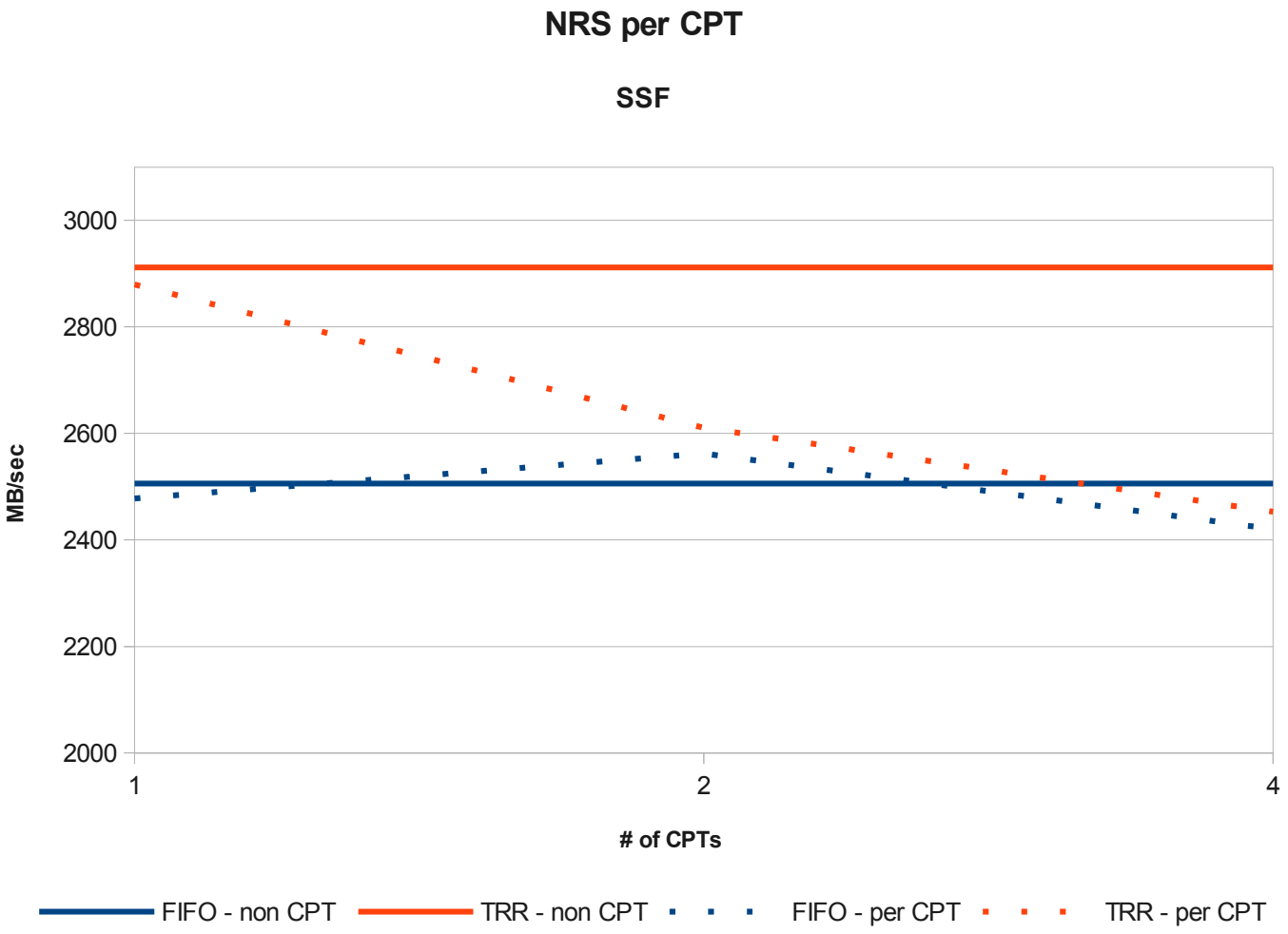
FPP Read - NRS per CPT



FFP Read - One NRS for all CPTs

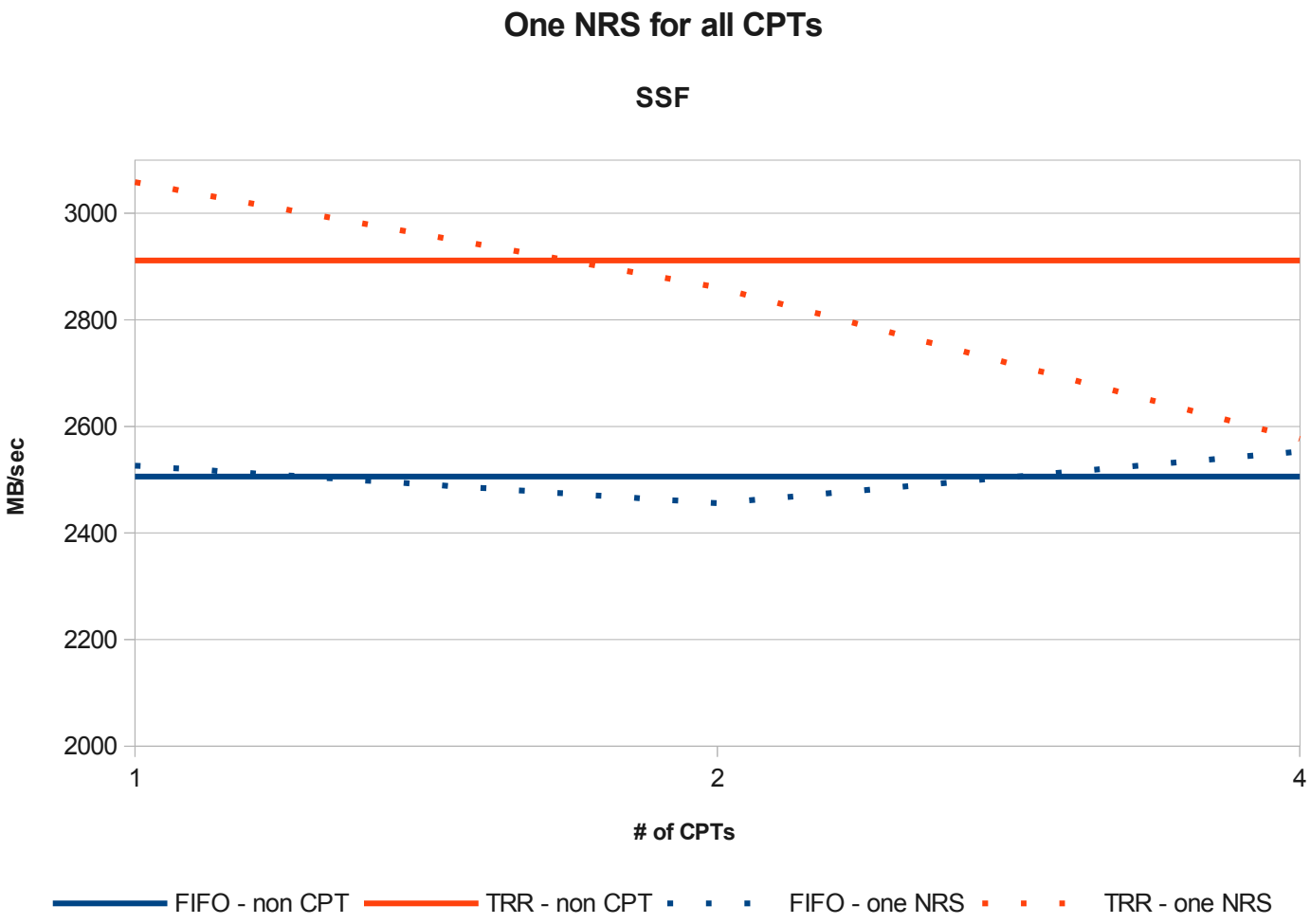


SSF Read - NRS per CPT



xyratex

SSF Read - One NRS for all CPTs



xyratex

Notes on ORR and TRR policies

- ORR and/or TRR may help improve:
  - Some generic read use cases
  - Small and/or random reads
  - Widely striped file reads
  - Backward reads
  - Cases in which OSTs are underutilized; this has not been tested yet
  - Reads by aligning writes
- ORR will need an LRU-based or similar method for object destruction; TRR much less so
- TRR and ORR should be less (if at all) beneficial on SSD-based OSTs

Increase read performance by aligning writes

- Possibly increase read performance by aligning writes
- Write performance takes a hit
- But this may be useful in read-mostly or read-important cases
- Quick, small scale test
  - Again 14 clients, 1 CS3000 (with 2 OSS), ost_io.threads_max = 128, stripped directories

Test	policy writing	policy reading	write (MB/s)	read (MB/s)
FPP	FIFO	FIFO	2013.72	2735.63
	ORR (log, 256)	FIFO	1074.25	3937.05*
	ORR (log, 256)	ORR (phys, 256)	1074.25	3966.07*
SSF	FIFO	FIFO	2094.56	2832.48
	ORR (log, 256)	FIFO	1115.28	3226.53
	ORR (log, 256)	ORR (phys, 256)	1115.28	3186.26

* value is >> quoted system maximum

Future tasks

- Need to test the policies at scale with different CPT configurations
  - Test on large NUMA servers
  - Perhaps with CPT-enabled libcfs_heap
  - Decide on the best SMP scaling-enabled adaptation
- Test with ZFS backend and OSD-restructured servers
  - Possibly update for ZFS
- Two or more policies working at the same time could be useful
- Merge PTLRPC service and NRS request stats
- NRS policies as separate kernel modules
- Is there a way to improve write performance using NRS?
- Investigate other possible applications

xyratex

Advancing Digital Storage Innovation

Thank you!

Nikitas Angelinas
nikitas_angelinas@xyratex.com