Advancing Digital Storage Innovation

Network Request Scheduler Scale Testing Results

Nikitas Angelinas
nikitas_angelinas@xyratex.com

Agenda

- NRS background
- Aim of test runs
- Tools used
- Test results
- Future tasks

NRS motivation

- Increased read and write throughput across the filesystem.
- Increased fairness amongst filesystem nodes, and better utilization of resources.
  - Clients.
  - OSTs.
  - Network.
- Deliberate and controlled unfairness amongst filesystem nodes; QoS semantics.
  - Client or export prioritization.
  - Guaranteed minimum and maximum throughput for a client.

Foreword

- NRS is a collaborative project between Whamcloud and Xyratex.
  - Code is at git://git.whamcloud.com/fs/lustre-dev.git repo, branch liang/b_nrs.
  - Jira ticket LU-398.
  - Is waiting for some large-scale testing.
- It allows the PTLRPC layer to reorder the servicing of incoming RPCs.
  - We are mostly interested in bulk I/O RPCs.
- Predominantly server-based, although the clients could play a part in some use cases.

NRS policies

- A binary heap data type is added to libcfs.
  - Used to implement prioritized queues of RPCs at the server.
  - Sorts large numbers of RPCs (10,000,000+) with minimal insertion/removal time.
- 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, with request ordering according to logical or physical offsets.
- TRR - Target Round Robin, RR over OSTs, with request ordering according to logical or physical offsets.
- Client prioritization policy (not yet implemented).
- QoS, or guaranteed availability policy (not yet implemented).

NRS features

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

Questions to be answered

- Any performance regressions for the NRS framework with FIFO policy?
- Scalability to a large number of clients?
- Effective implementation of the algorithms?
- Are other policies besides FIFO useful?
  - A given policy may aid performance in particular situations, while hindering performance in other situations.
  - A given policy may also benefit more generic aspects of the filesystem workload.
- Provide quantified answers to the above via a series of tests performed at large scale.

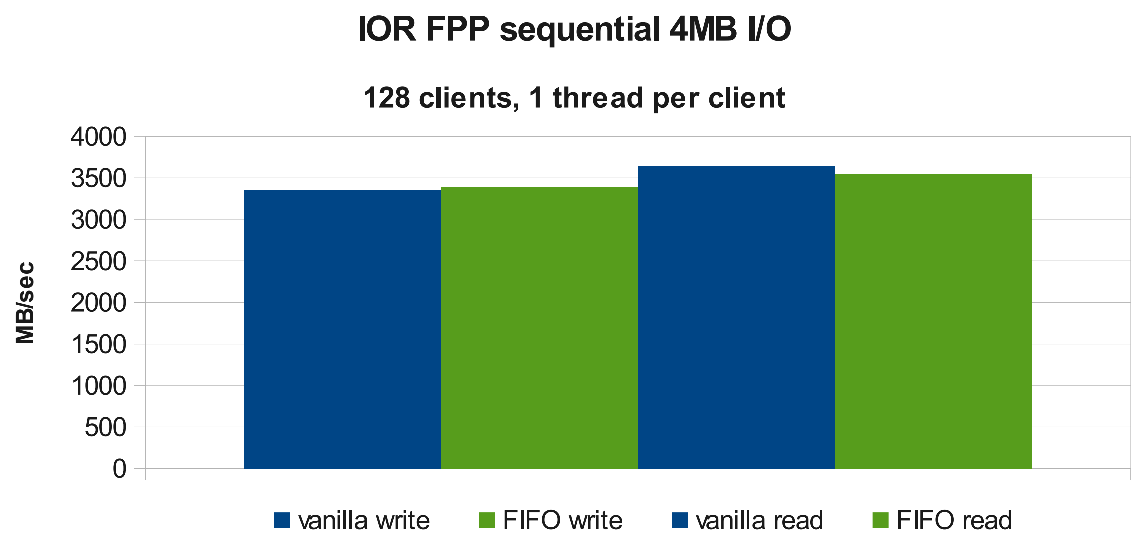
Benchmarking environment

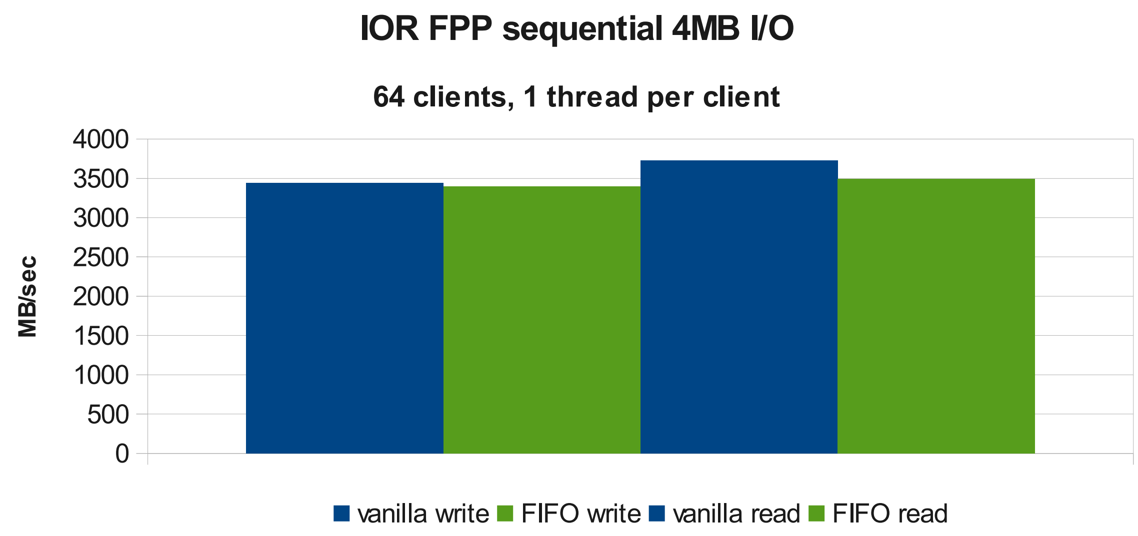
- NRS code rebased on the same Git commit as vanilla; apples vs apples.
- IOR for SSF and FPP runs of sequential I/O tests.
- mdtest for file and directory operations metadata performance.
- IOzone in clustered mode.
- Multi-client test script using groups of dd processes.
- 1 Xyratex CS3000; 2 x OSS, 4 x OSTs each.
- 10 - 128 physical clients, depending on test case and resource availability.
- Infiniband QDR fabric.
- Larger scale tests performed at University of Cambridge; smaller scale tests performed in-house.

Performance regression with FIFO policy

- IOR SSF and FPP, and mdtest runs.
- Looking for major performance regressions; minor performance regressions would be hidden by the variance between test runs.
- So these tests aim to give us indications, but not definite assurance.
- IOR FPP: IOR -v -a POSIX -i3 -g -e -w -W -r -b 16g -C -t 4m -F -o /mnt/lustre/testfile.fpp -O lustreStripeCount=1 .
- IOR SSF: IOR -v -a POSIX -i3 -g -e -w -W -r -b 16g -C -t 4m -o /mnt/lustre/testfile.ssf -O lustreStripeCount=-1 .
- mdtest: mdtest -u -d /mnt/lustre/mdtest{1-128} -n 32768 -i 3 .

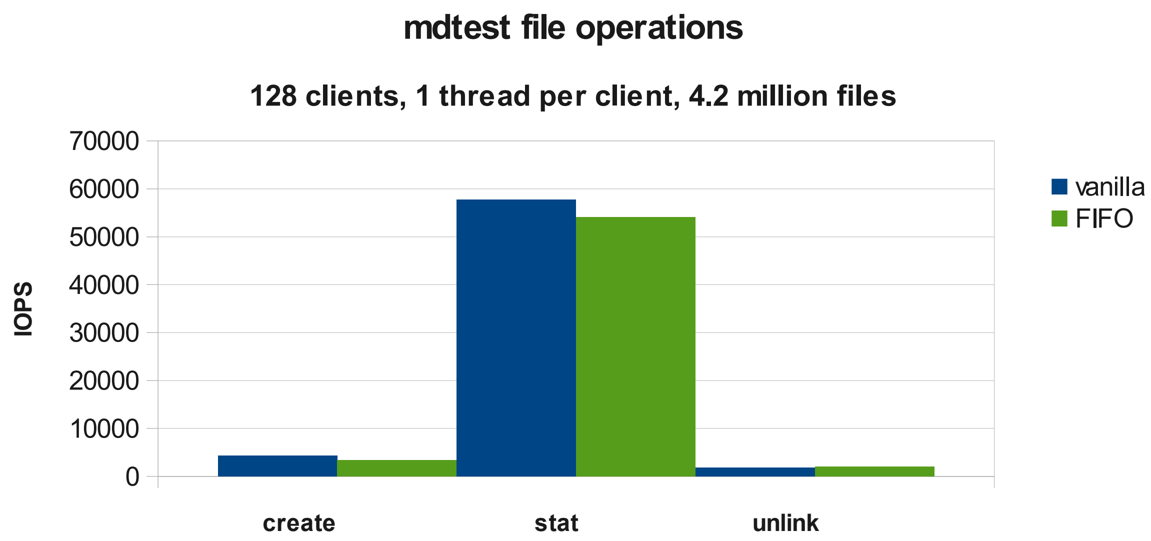
IOR FPP regression testing

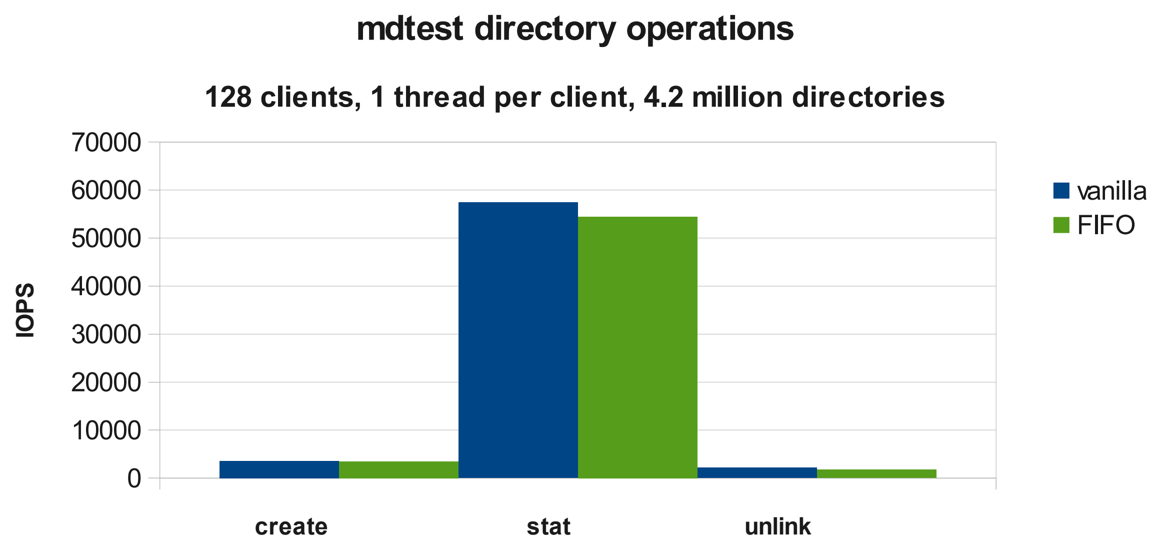




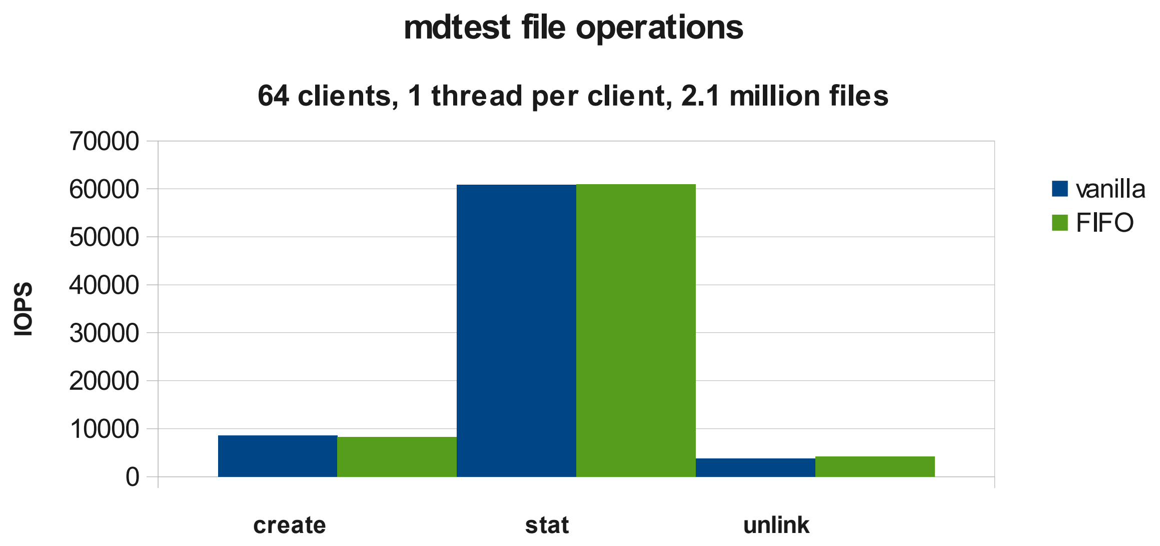
IOR SSF regression testing

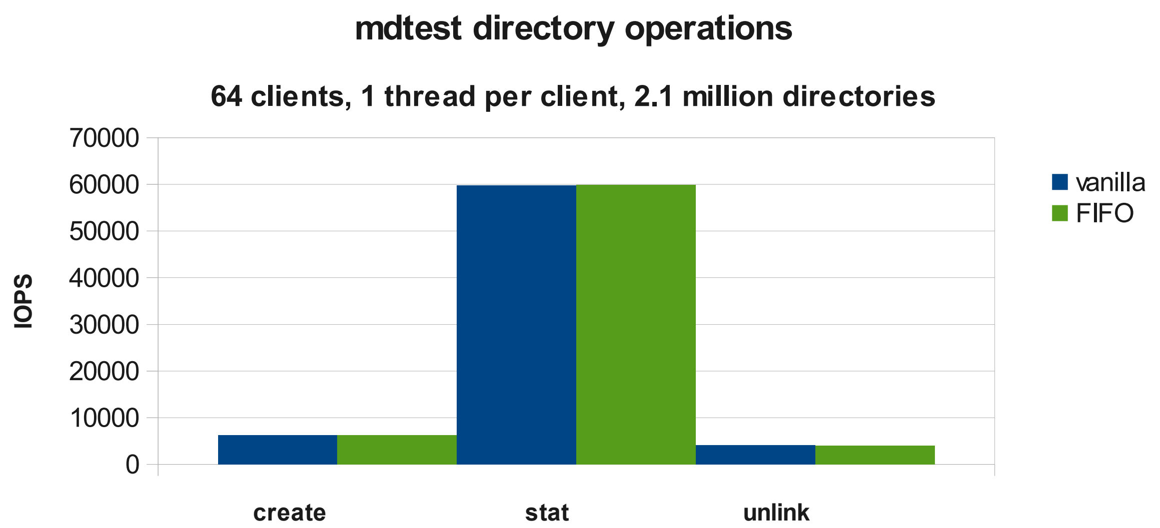
mdtest file and dir ops regression testing



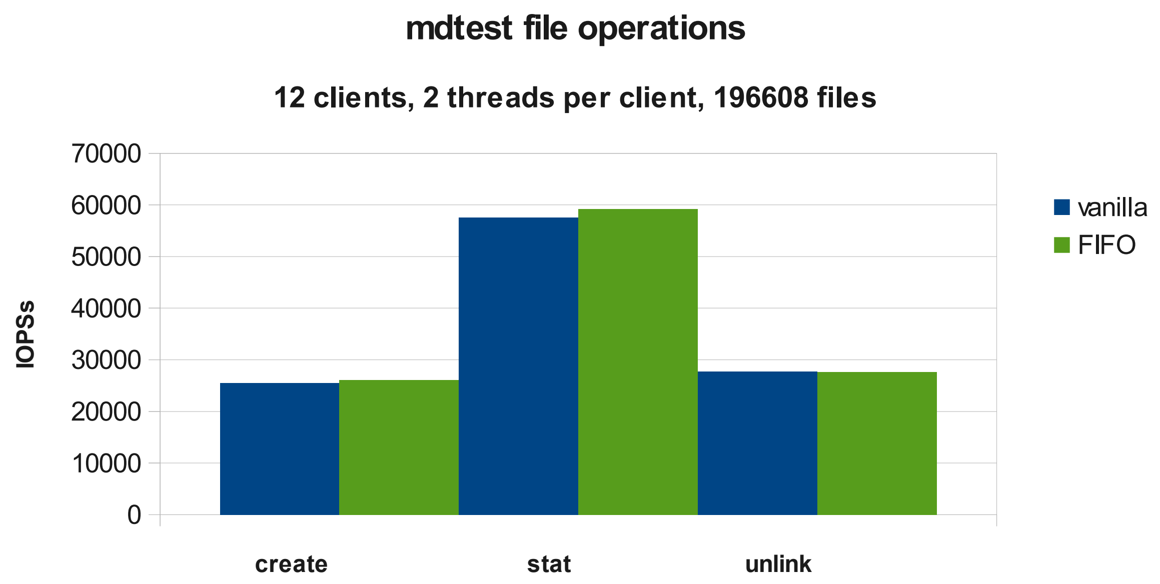


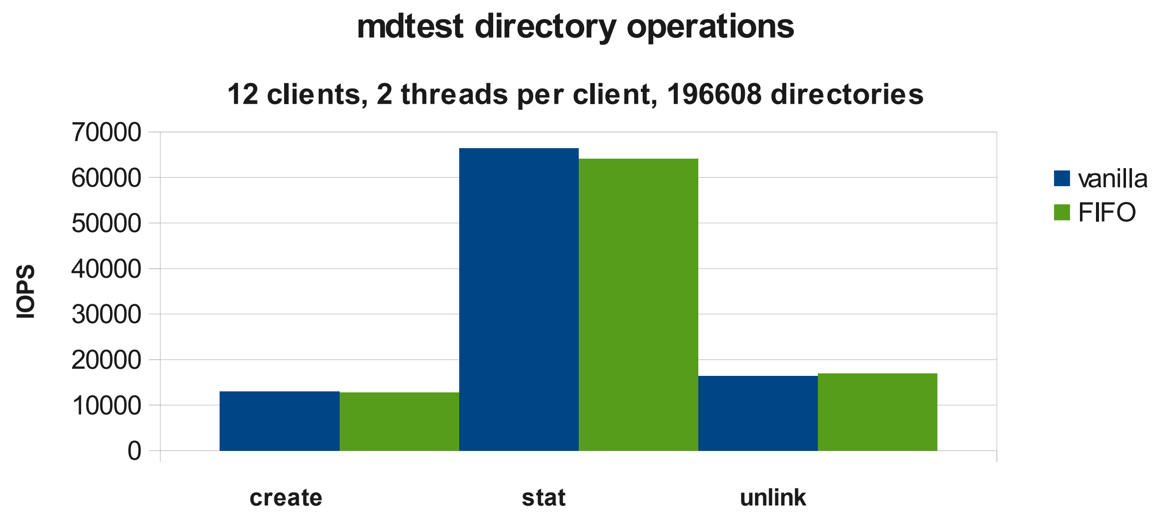
mdtest file and dir ops regression testing





mdtest file and dir ops regression testing





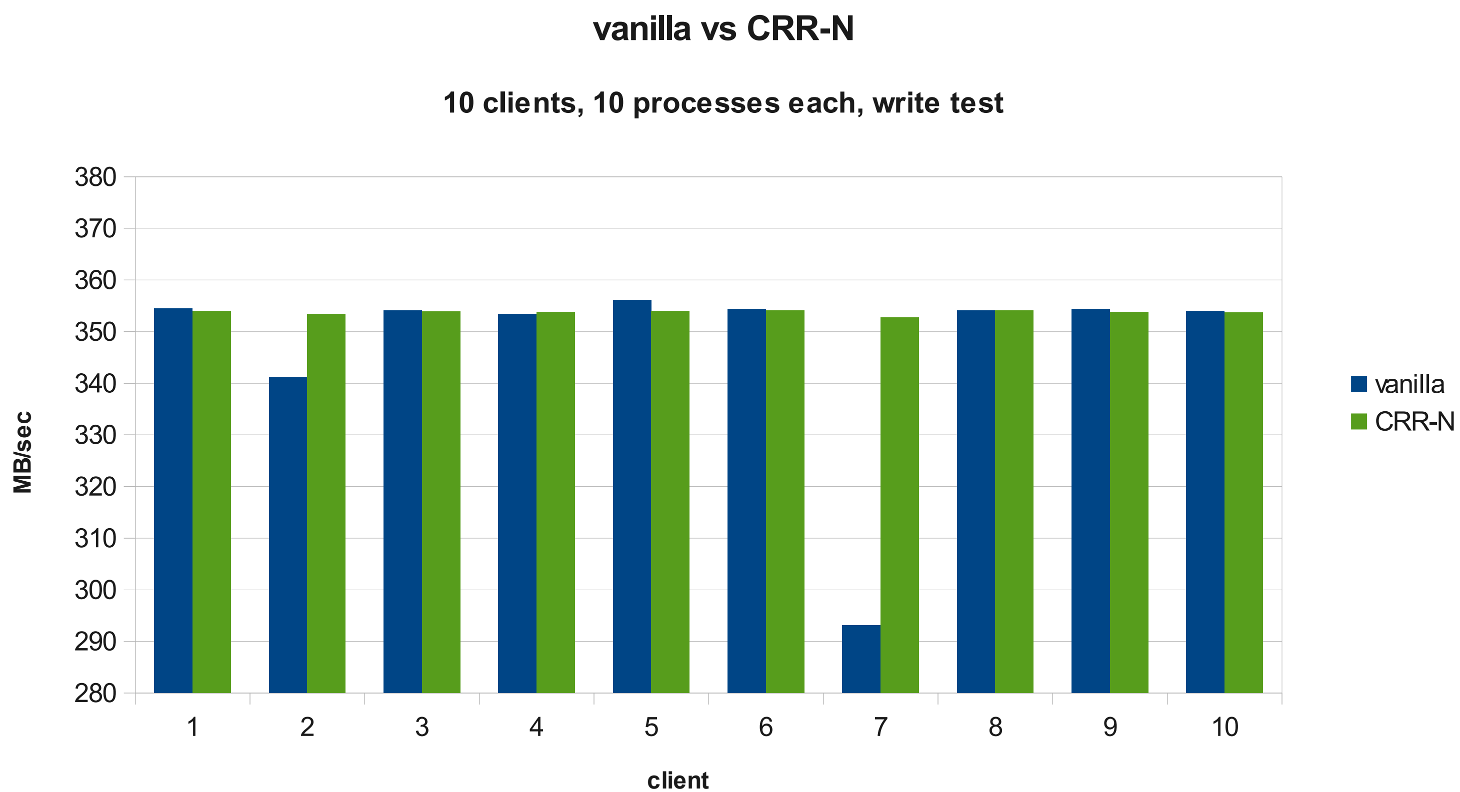
CRR-N dd-based investigation

- Groups of dd processes.
  - Read test: dd if=/mnt/lustre/dd_client*/BIGFILE* of=/dev/null bs=1M .
  - Write test: dd if=/dev/zero of=/mnt/lustre/dd_client*/outfile* bs=1M .
- Two series of test runs:
  - 10 clients with 10 dd processes each.
  - 9 clients with 11 dd processes, 1 client with 1 dd process.
- Observe the effect of NRS CRR-N vs vanilla for the above test runs.
  - Measure throughput at each client.
  - Calculate standard deviation of throughput.

CRR-N brw RPC distribution (dd test, 14 clients)

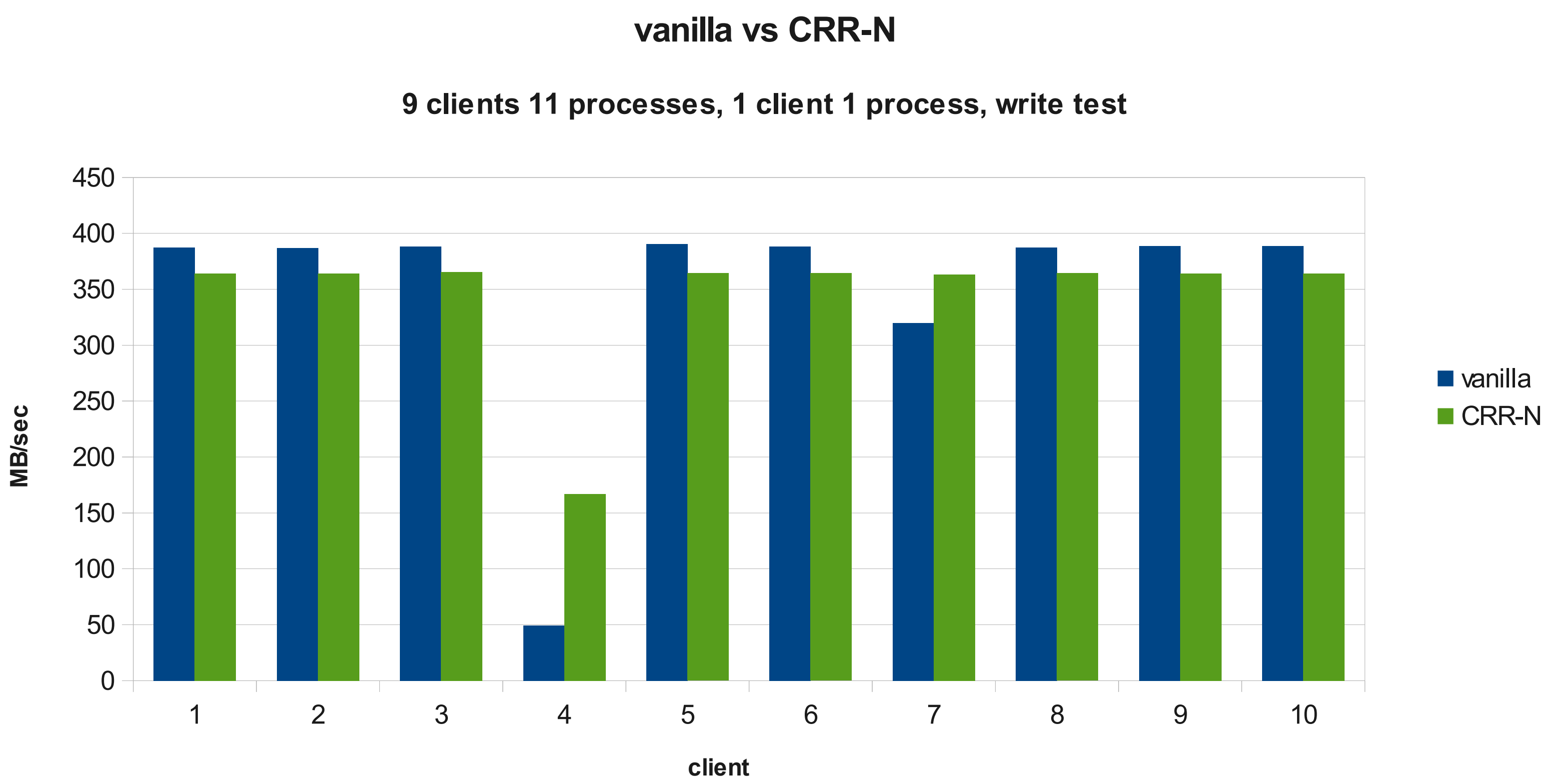
```
NRS start crr2 request from 12345-172.18.1.128@o2ib, round 6975, seq: 85593
NRS start crr2 request from 12345-172.18.1.122@o2ib, round 6975, seq: 85682
NRS start crr2 request from 12345-172.18.1.124@o2ib, round 6975, seq: 85686
NRS start crr2 request from 12345-172.18.1.127@o2ib, round 6975, seq: 85734
NRS start crr2 request from 12345-172.18.1.123@o2ib, round 6975, seq: 85744
NRS start crr2 request from 12345-172.18.1.126@o2ib, round 6975, seq: 85757
NRS start crr2 request from 12345-172.18.1.118@o2ib, round 6975, seq: 85794
NRS start crr2 request from 12345-172.18.1.117@o2ib, round 6975, seq: 85839
NRS start crr2 request from 12345-172.18.1.131@o2ib, round 6975, seq: 85861
NRS start crr2 request from 12345-172.18.1.121@o2ib, round 6975, seq: 85923
NRS start crr2 request from 12345-172.18.1.129@o2ib, round 6975, seq: 85969
NRS start crr2 request from 12345-172.18.1.125@o2ib, round 6975, seq: 85981
NRS start crr2 request from 12345-172.18.1.119@o2ib, round 6976, seq: 85482
NRS start crr2 request from 12345-172.18.1.120@o2ib, round 6976, seq: 85495
NRS start crr2 request from 12345-172.18.1.128@o2ib, round 6976, seq: 85637
NRS start crr2 request from 12345-172.18.1.122@o2ib, round 6976, seq: 85683
NRS start crr2 request from 12345-172.18.1.124@o2ib, round 6976, seq: 85687
NRS start crr2 request from 12345-172.18.1.127@o2ib, round 6976, seq: 85735
NRS start crr2 request from 12345-172.18.1.123@o2ib, round 6976, seq: 85745
NRS start crr2 request from 12345-172.18.1.126@o2ib, round 6976, seq: 85761
NRS start crr2 request from 12345-172.18.1.117@o2ib, round 6976, seq: 85840
NRS start crr2 request from 12345-172.18.1.131@o2ib, round 6976, seq: 85866
NRS start crr2 request from 12345-172.18.1.118@o2ib, round 6976, seq: 85882
NRS start crr2 request from 12345-172.18.1.121@o2ib, round 6976, seq: 85926
NRS start crr2 request from 12345-172.18.1.129@o2ib, round 6976, seq: 85970
NRS start crr2 request from 12345-172.18.1.125@o2ib, round 6976, seq: 86030
```

dd write test - 10 clients, each with 10 processes



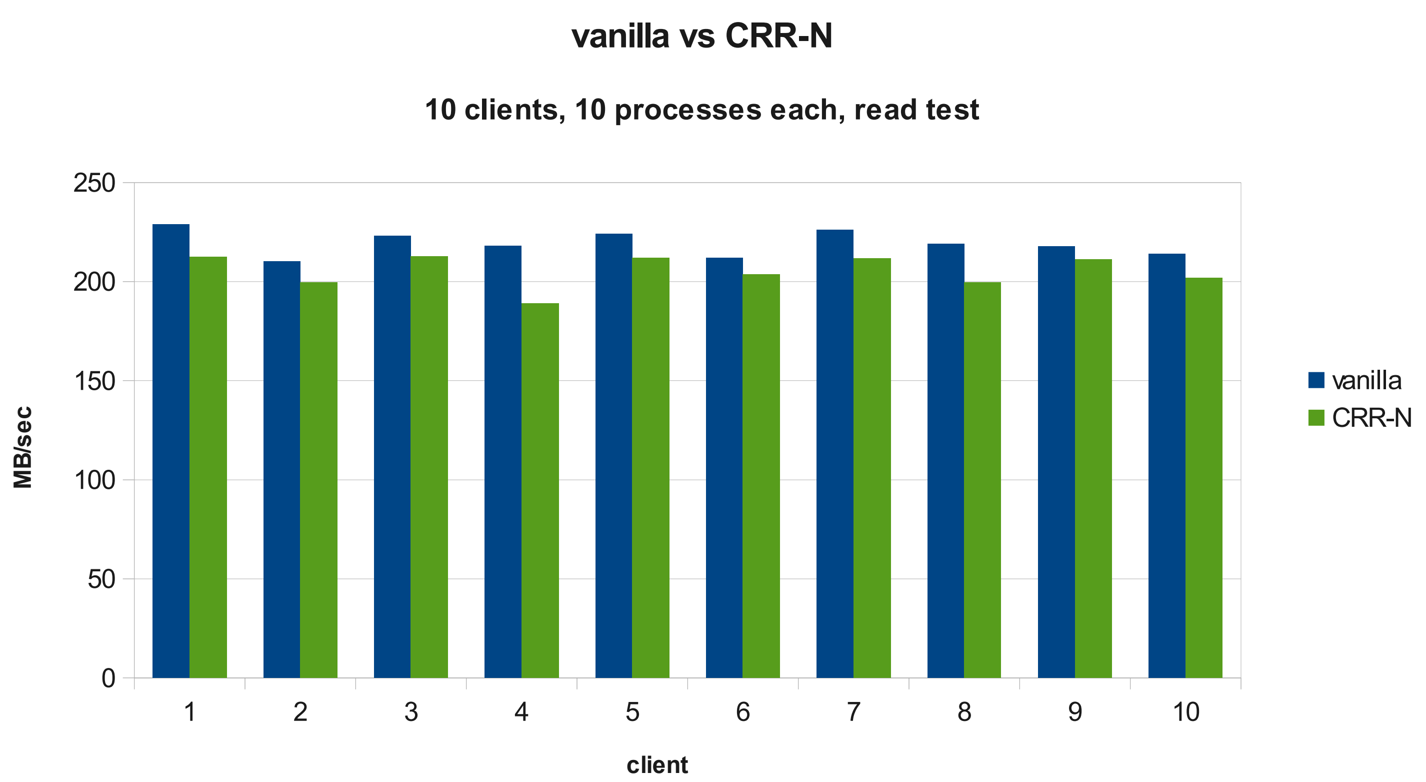
handler	stdev	throughput
vanilla	19.361 MB/sec	3469 MB/sec
CRR-N	0.425 MB/sec	3537.5 MB/sec

dd write test – 9 clients with 11 procs, client 4 with 1 proc



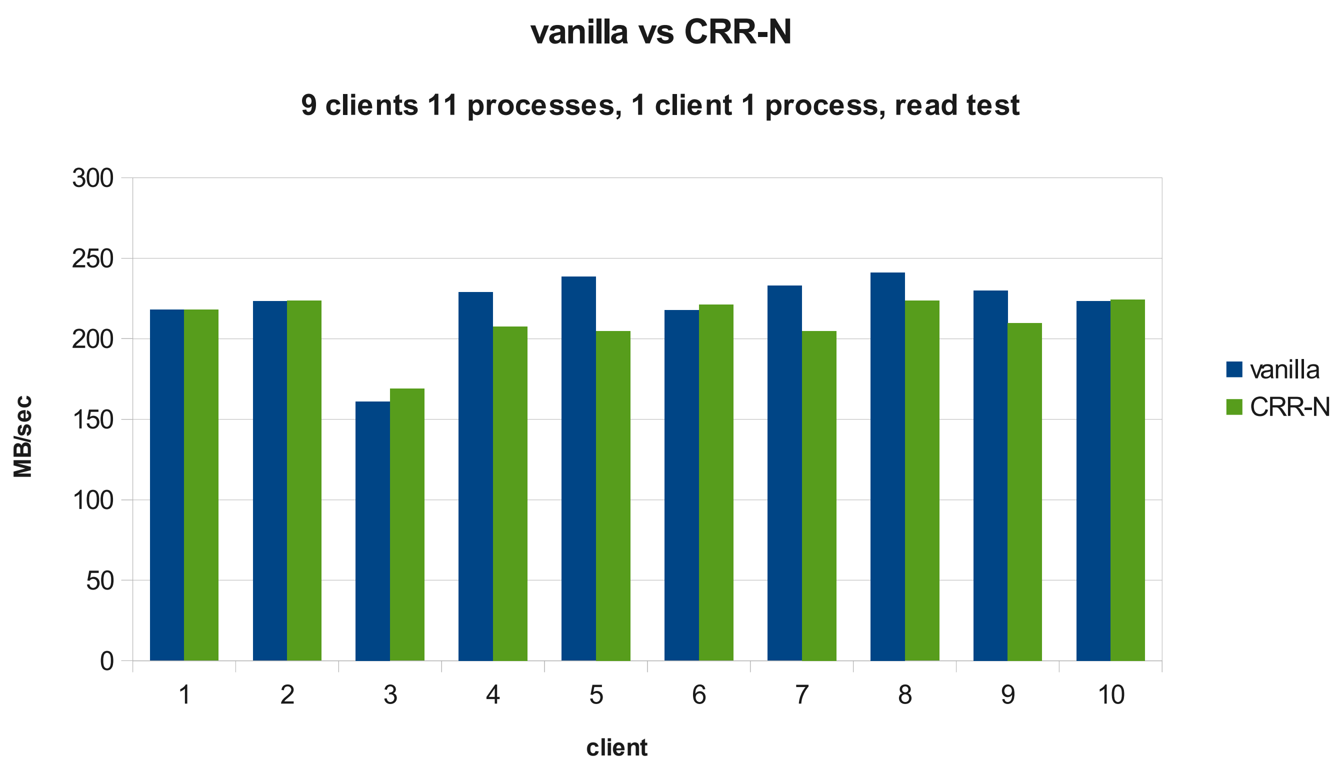
handler	stdev (client 4 excluded)	client 4	throughput
vanilla	22.756 MB/sec	49.2 MB/sec	3473.9 MB/sec
CRR-N	0.491 MB/sec	167 MB/sec	3444 MB/sec

dd read test - 10 clients, each with 10 processes, 128 threads



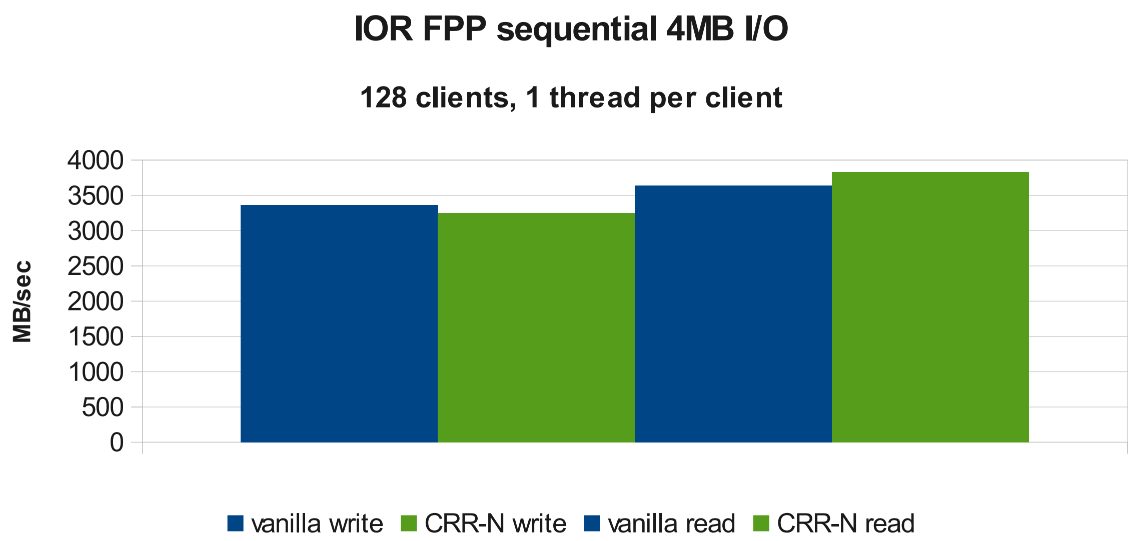
handler	stdev	throughput
vanilla	6.156 MB/sec	2193.8 MB/sec
CRR-N	7.976 MB/sec	2054.6 MB/sec

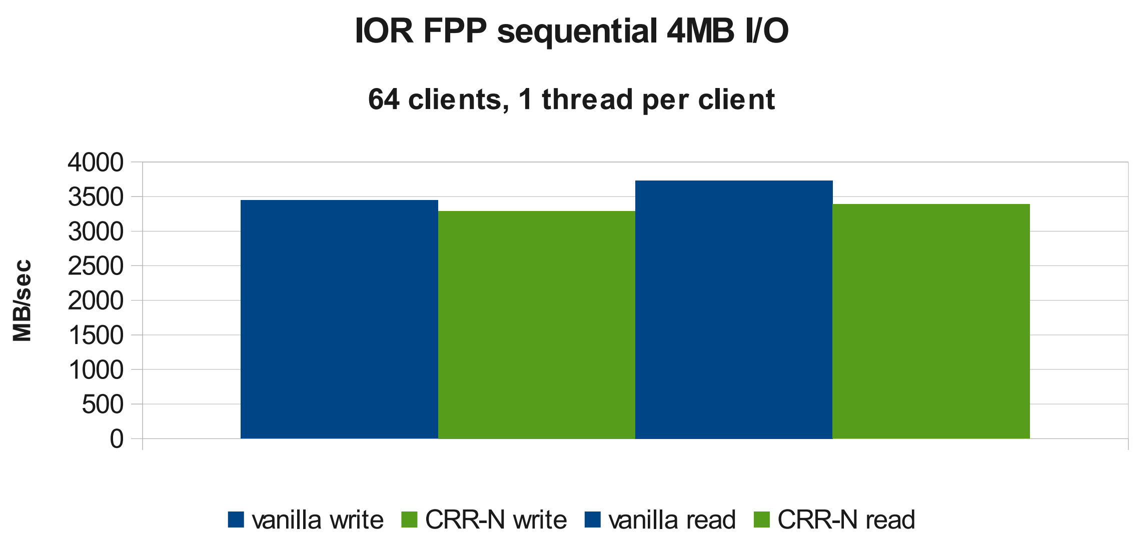
dd read test - 9 clients with 11 procs, client 3 with 1 proc, 128 threads



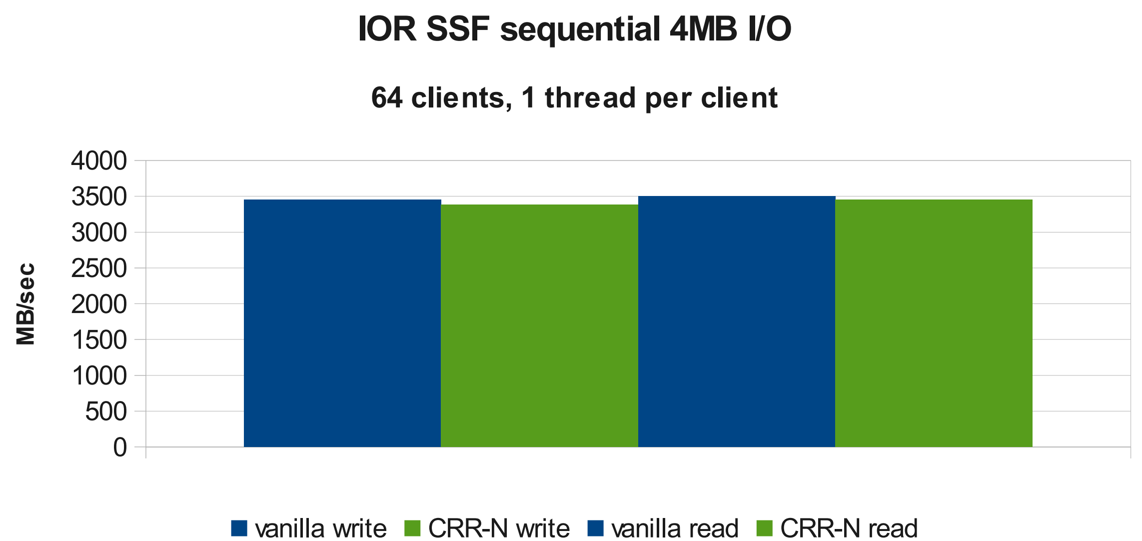
handler	stdev (client 3 excluded)	client 3	throughput
vanilla	7.490 MB/sec	161 MB/sec	2229.2 MB/sec
CRR-N	8.455 MB/sec	169 MB/sec	2106.6 MB/sec

IOR FPP - vanilla vs NRS with CRR-N policy





IOR SSF - vanilla vs NRS with CRR-N policy



CRR-N comments

- CRR-N causes a significant lowering of the stdev of write throughput.
  - i.e. it 'evens things out'.
  - Many users will want this.
- CRR-N shows a negative effect on dd test read operations, but IOR regression tests are fine.
  - Worst case, reads could be routed to FIFO or other policy.
- CRR-N may improve compute cluster performance when used with real jobs that do some processing.
- No performance regressions on IOR tests.
  - Confidence to deploy in real clusters and get real-world feedback.
- Future testing task is to see if these results scale.

ORR/TRR policies

- ORR serves bulk I/O RPCs (only OST_READs by default) in a Round Robin manner over available backend-fs objects.
  - RPCs are grouped in per-object groups of 'RR quantum' size; lprocfs tunable.
  - Sorted within each group by logical of physical disk offset.
  - Physical offsets are calculated using extent information obtained via fiemap.
- TRR performs the same scheduling, but in a Round Robin manner over available OSTs.
- The main aim is to minimize drive seek operations, thus increasing read performance.
- TRR should be able to help in cases where an OST is underutilized; this was not straightforward to test.

ORR (phys, 8) brw RPC distribution (IOR FPP test)

```
NRS start orr request for object with ID 3045868 from OST with index 3, with round 3201
NRS start orr request for object with ID 3045868 from OST with index 3, with round 3201
NRS start orr request for object with ID 3045868 from OST with index 3, with round 3201
NRS start orr request for object with ID 3045868 from OST with index 3, with round 3201
NRS start orr request for object with ID 3045868 from OST with index 3, with round 3201
NRS start orr request for object with ID 3045868 from OST with index 3, with round 3201
NRS start orr request for object with ID 3045868 from OST with index 3, with round 3201
NRS start orr request for object with ID 3045868 from OST with index 3, with round 3201
NRS start orr request for object with ID 2969230 from OST with index 1, with round 3202
NRS start orr request for object with ID 2969230 from OST with index 1, with round 3202
NRS start orr request for object with ID 2969230 from OST with index 1, with round 3202
NRS start orr request for object with ID 2969230 from OST with index 1, with round 3202
NRS start orr request for object with ID 2969230 from OST with index 1, with round 3202
NRS start orr request for object with ID 2969230 from OST with index 1, with round 3202
NRS start orr request for object with ID 2969230 from OST with index 1, with round 3202
NRS start orr request for object with ID 2969230 from OST with index 1, with round 3202
NRS start orr request for object with ID 2950395 from OST with index 2, with round 3203
NRS start orr request for object with ID 2950395 from OST with index 2, with round 3203
NRS start orr request for object with ID 2950395 from OST with index 2, with round 3203
NRS start orr request for object with ID 2950395 from OST with index 2, with round 3203
NRS start orr request for object with ID 2950395 from OST with index 2, with round 3203
NRS start orr request for object with ID 2950395 from OST with index 2, with round 3203
NRS start orr request for object with ID 2950395 from OST with index 2, with round 3203
NRS start orr request for object with ID 2947110 from OST with index 0, with round 3185
NRS start orr request for object with ID 2950395 from OST with index 2, with round 3203
NRS start orr request for object with ID 2969231 from OST with index 1, with round 3204
```

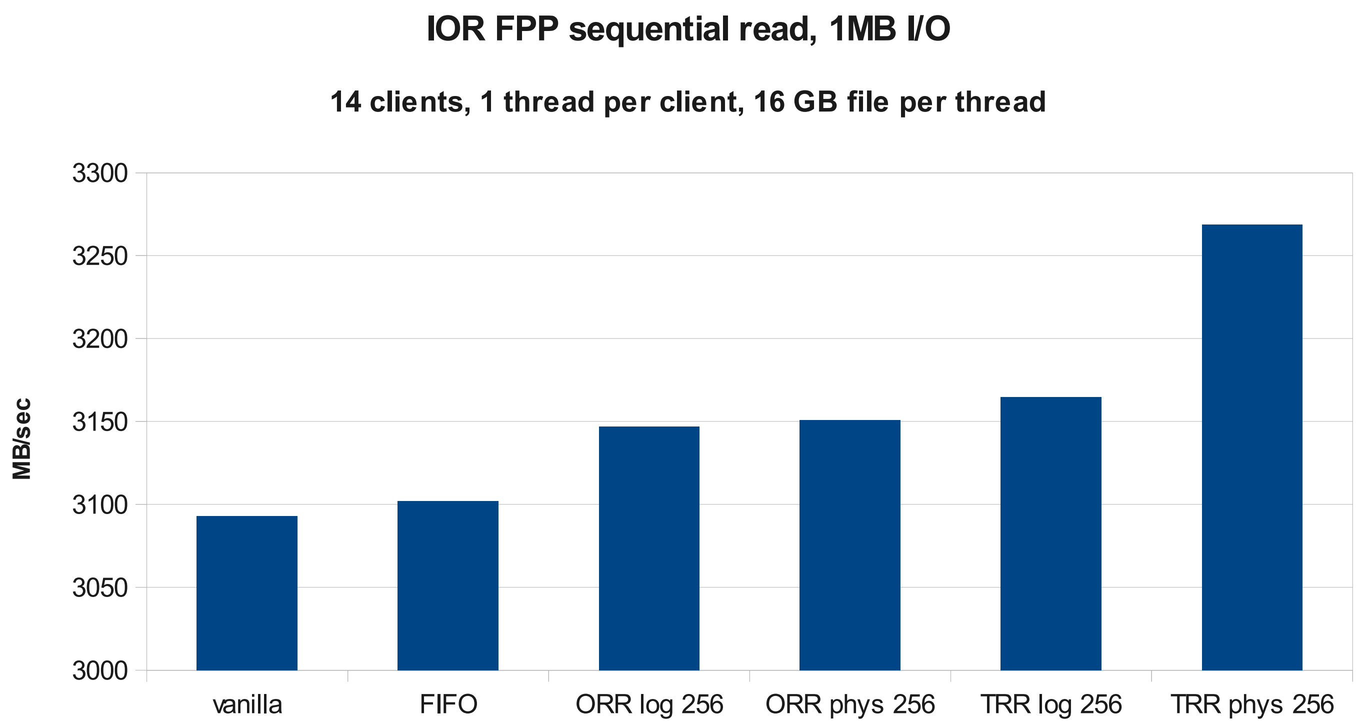
TRR (phys, 8) brw RPC distribution (IOR FPP test)

```
NRS start orr request for object with ID 0 from OST with index 1, with round 2654
NRS start orr request for object with ID 0 from OST with index 1, with round 2654
NRS start orr request for object with ID 0 from OST with index 1, with round 2654
NRS start orr request for object with ID 0 from OST with index 1, with round 2654
NRS start orr request for object with ID 0 from OST with index 1, with round 2654
NRS start orr request for object with ID 0 from OST with index 1, with round 2654
NRS start orr request for object with ID 0 from OST with index 1, with round 2654
NRS start orr request for object with ID 0 from OST with index 1, with round 2654
NRS start orr request for object with ID 0 from OST with index 3, with round 2655
NRS start orr request for object with ID 0 from OST with index 3, with round 2655
NRS start orr request for object with ID 0 from OST with index 3, with round 2655
NRS start orr request for object with ID 0 from OST with index 3, with round 2655
NRS start orr request for object with ID 0 from OST with index 3, with round 2655
NRS start orr request for object with ID 0 from OST with index 3, with round 2655
NRS start orr request for object with ID 0 from OST with index 3, with round 2655
NRS start orr request for object with ID 0 from OST with index 3, with round 2655
NRS start orr request for object with ID 0 from OST with index 0, with round 2656
NRS start orr request for object with ID 0 from OST with index 0, with round 2656
NRS start orr request for object with ID 0 from OST with index 0, with round 2656
NRS start orr request for object with ID 0 from OST with index 0, with round 2656
NRS start orr request for object with ID 0 from OST with index 1, with round 2657
NRS start orr request for object with ID 0 from OST with index 1, with round 2657
NRS start orr request for object with ID 0 from OST with index 1, with round 2657
NRS start orr request for object with ID 0 from OST with index 1, with round 2657
NRS start orr request for object with ID 0 from OST with index 1, with round 2657
NRS start orr request for object with ID 0 from OST with index 1, with round 2657
NRS start orr request for object with ID 0 from OST with index 1, with round 2657
NRS start orr request for object with ID 0 from OST with index 1, with round 2657
```

ORR/TRR policy tests

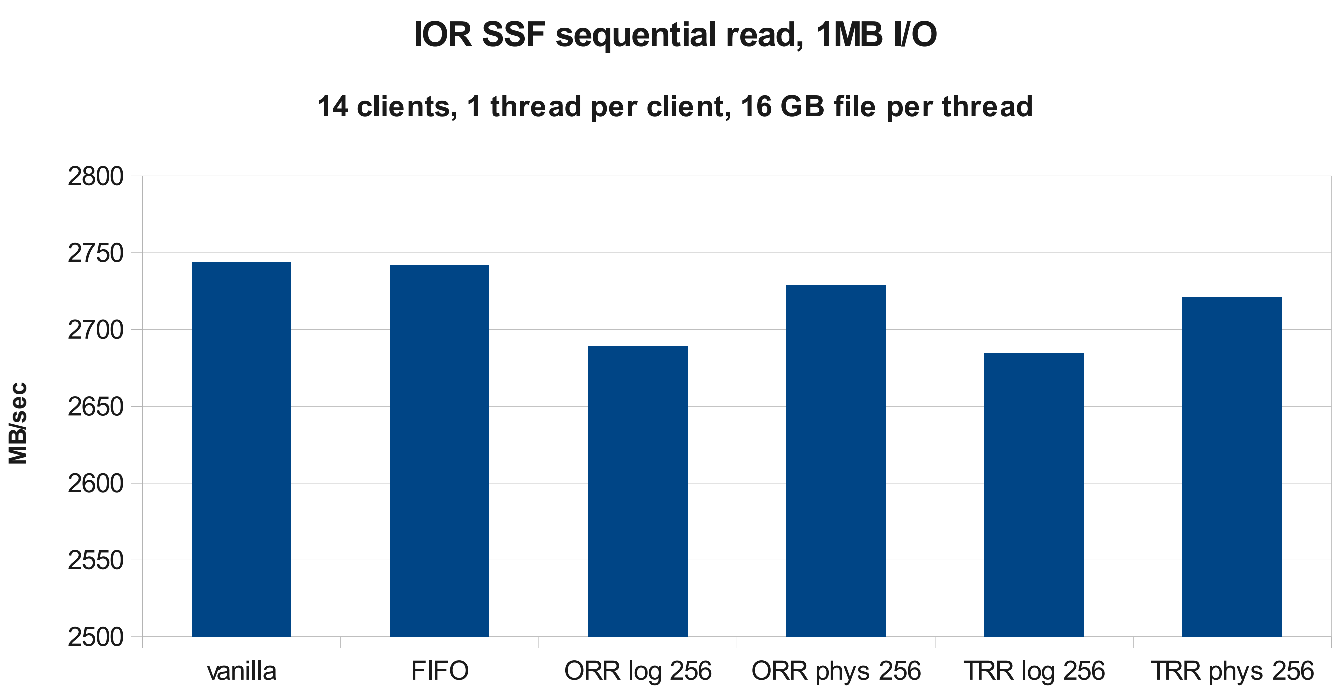
- Using IOR to perform read tests; each IOR process reads 16 GB of data.
    - Kernel caches cleared between reads.
- Performance is compared with vanilla and NRS FIFO for different TRR/ORR policy parameters.
- Tests with 1 process per client and 8 processes per client.
- Only 14 clients, read operations generate few RPCs.
    - ost_io.threads_max=128 on both OSS nodes.
- The OSS nodes are not totally saturated with this number of clients.

ORR/TRR policy IOR FPP read



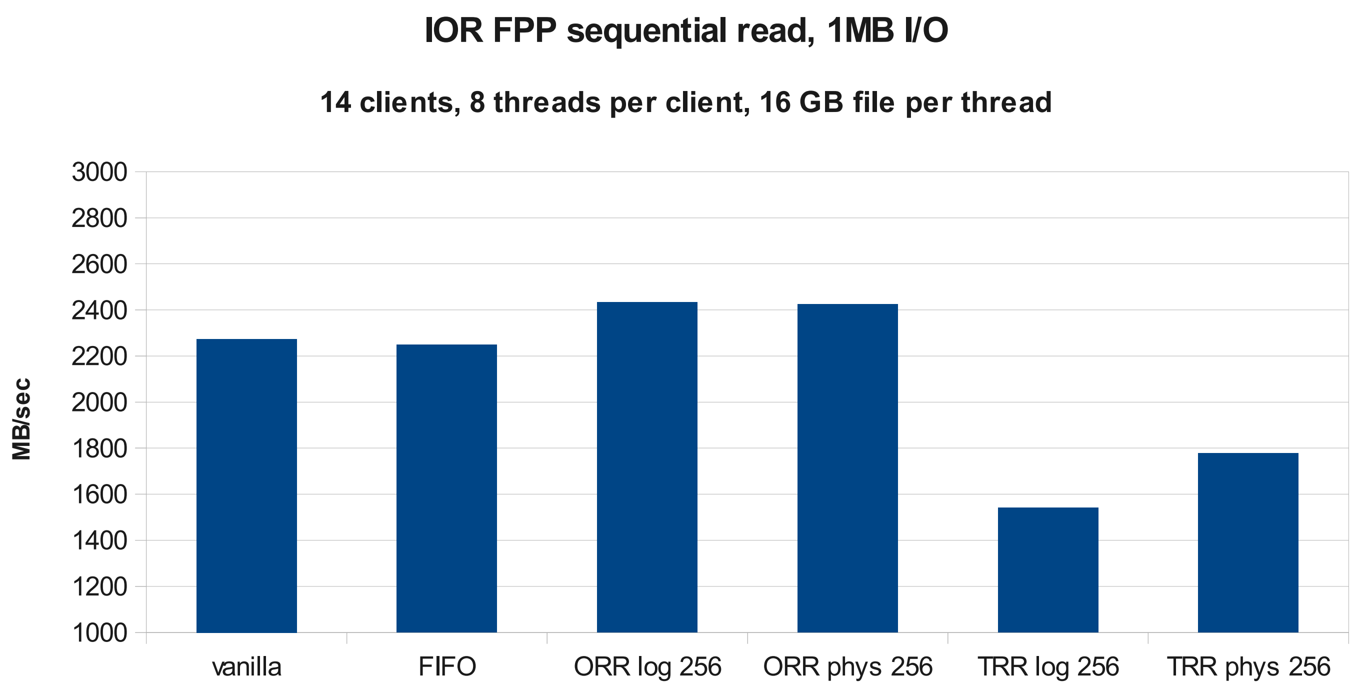
vanilla:	3092.91 MB/sec
FIFO:	3102.17 MB/sec
ORR log 256:	3146.97 MB/sec
ORR phys 256:	3150.86 MB/sec
TRR log 256:	3164.66 MB/sec
TRR phys 256:	3268.98 MB/sec

ORR/TRR policy IOR SSF read



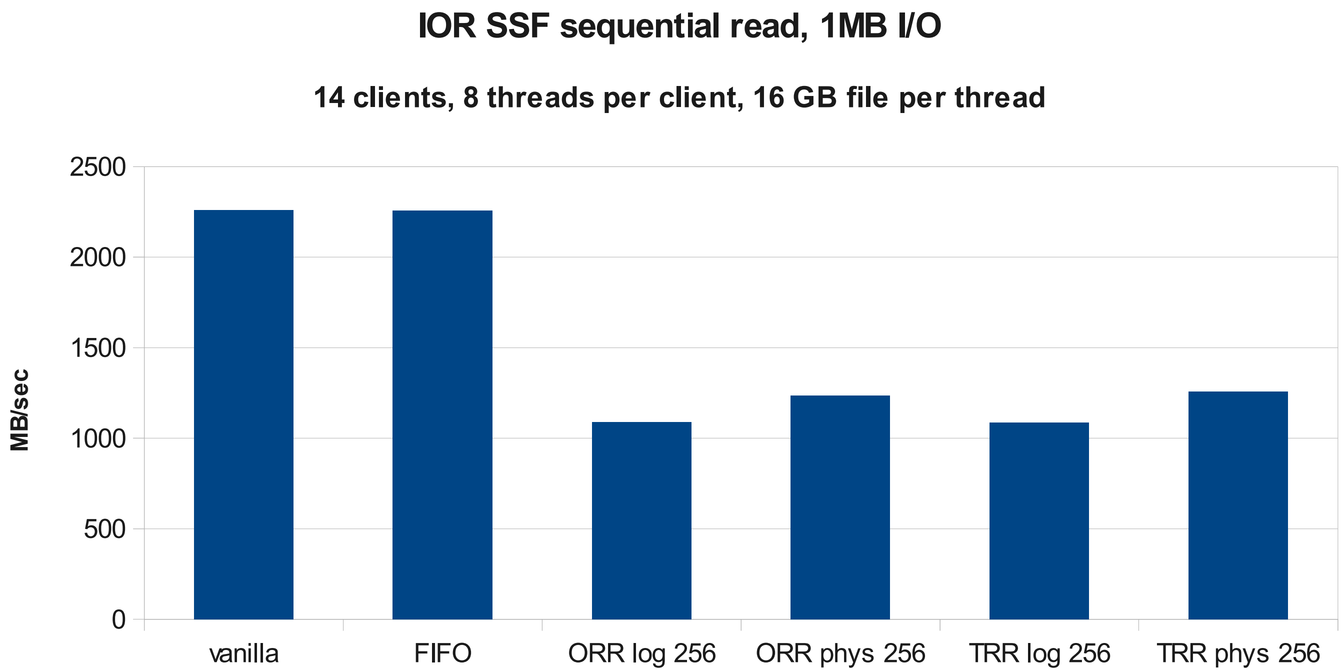
vanilla:	2744.15 MB/sec
FIFO:	2741.78 MB/sec
ORR log 256:	2689.12 MB/sec
ORR phys 256:	2728.89 MB/sec
TRR log 256:	2684.42 MB/sec
TRR phys 256:	2720.96 MB/sec

ORR/TRR policy IOR FPP read



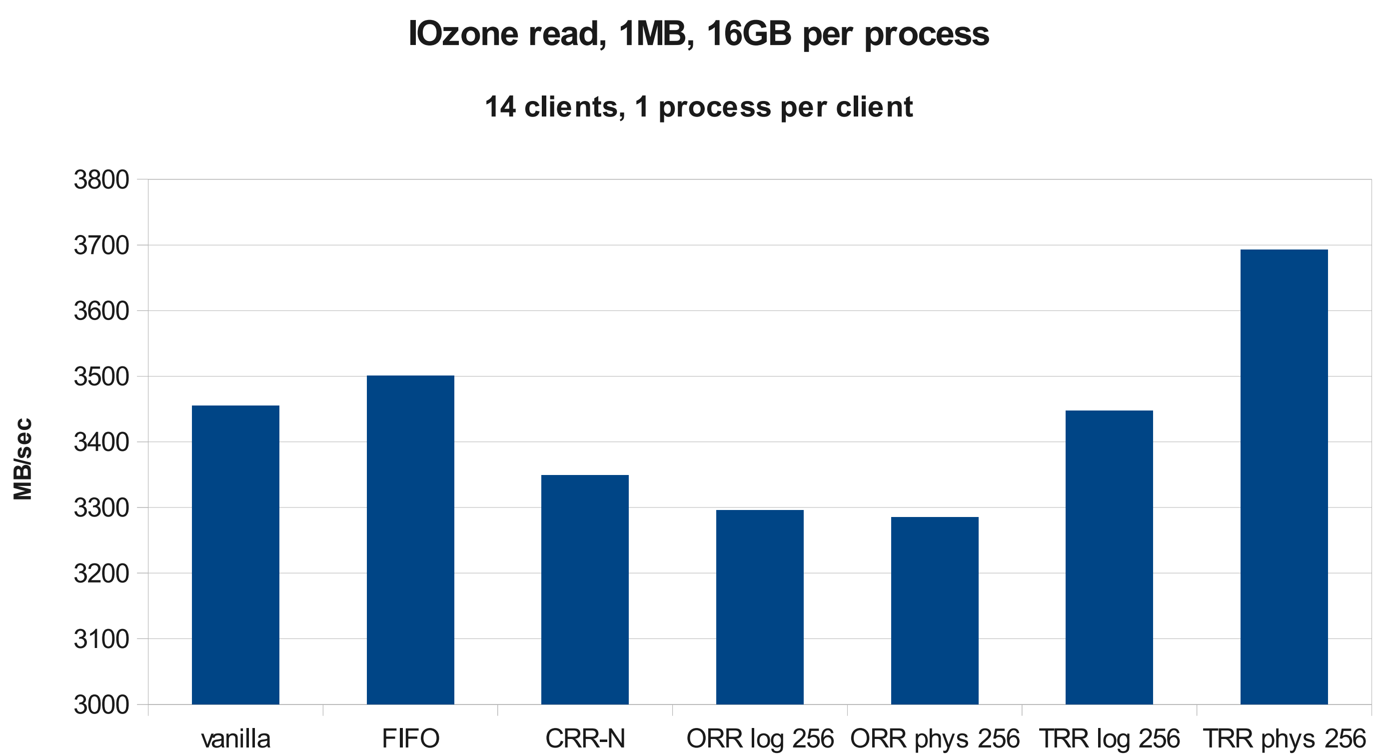
vanilla:	2274.55 MB/sec
FIFO:	2248.62 MB/sec
ORR log 256:	2432.78 MB/sec
ORR phys 256:	2424.69 MB/sec
TRR log 256:	1540.82 MB/sec
TRR phys 256:	1778.56 MB/sec

ORR/TRR policy IOR SSF read



vanilla:	2260.37 MB/sec
FIFO:	2257.9 MB/sec
ORR log 256:	1089.385 MB/sec
ORR phys 256:	1236.72 MB/sec
TRR log 256:	1086.18 MB/sec
TRR phys 256:	1258.907 MB/sec

IOzone read test – all policies



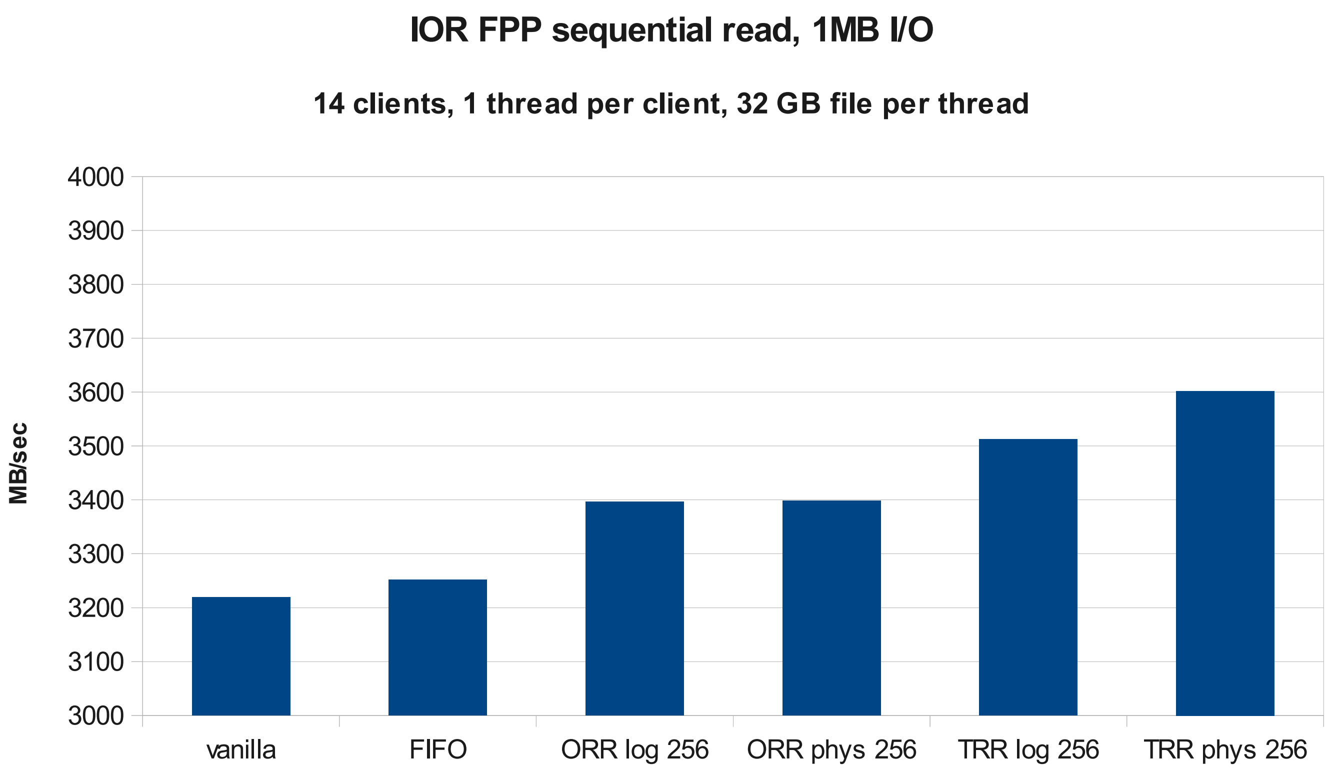
IOzone read test – throughput per process

14 clients, 1 process per client		
handler	min (MB/sec)	max (MB/sec)
vanilla	190.18	606.87
FIFO	191.51	618.05
CRR-N	188.79	513.87
ORR (log, 256)	198.8	425.27
ORR (phys, 256)	198.8	418.85
TRR (log, 256)	208.48	476.55
TRR (phys, 256)	217.56	488.25

ORR/TRR policy tests, large readahead

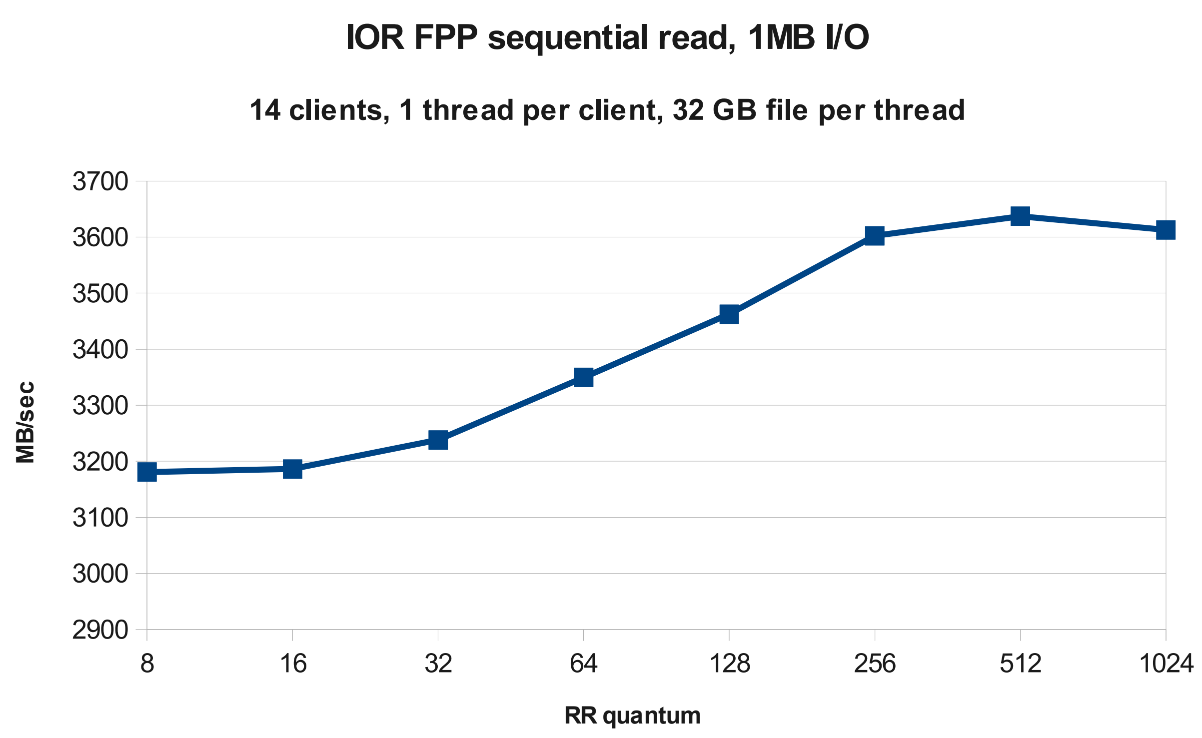
- Only 14 clients, for 512 ost_io threads tests, increase number of RPCs by:
  - max_read_ahead_mb=256
  - max_read_ahead_per_file_mb=256
- These lead to curious results.
    - Tests were without the LU-983 fix for readahead.

ORR/TRR policy IOR FPP read, large readahead



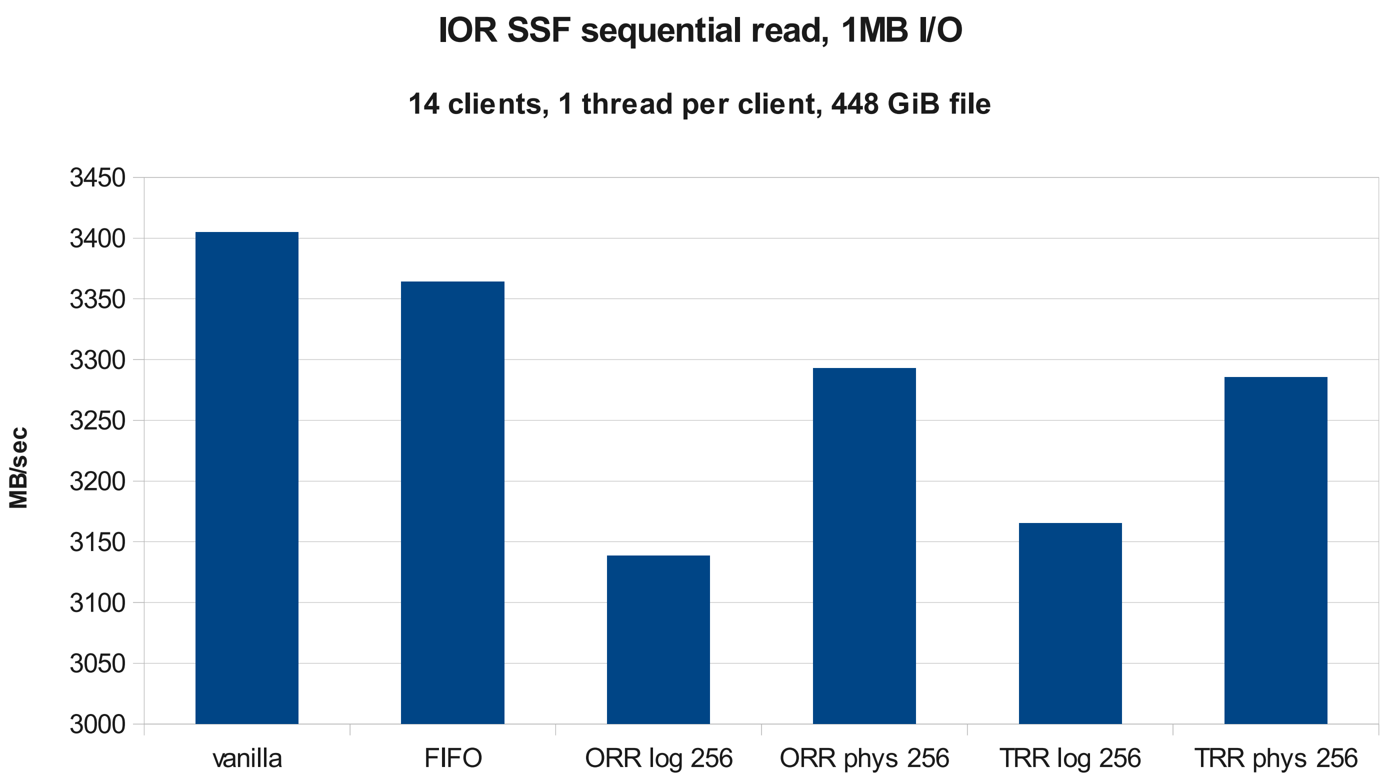
vanilla:	3219.73 MB/sec
FIFO:	3252.52 MB/sec
ORR log 256:	3396.07 MB/sec
ORR phys 256:	3398.86 MB/sec
TRR log 256:	3512.55 MB/sec
TRR phys 256:	3602.22 MB/sec

TRR (physical, 256) IOR FPP read, large readahead



- Performance is highest at a ~ 512 quantum size.
- The exact number may vary between workloads.

ORR/TRR IOR SSF read test, large readahead



vanilla:	3404.8 MB/sec
FIFO:	3364.2 MB/sec
ORR log 256:	3138.4 MB/sec
ORR phys 256:	3293.1 MB/sec
TRR log 256:	3165.3 MB/sec
TRR phys 256:	3285.5 MB/sec

Notes on ORR and TRR policies

- TRR/ORR increase performance in some test cases, but decrease it in others.
- TRR/ORR may improve the performance of small and/or random reads.
    - Random reads produce a small number of RPCs with few clients, so this was not tested.
- TRR may improve the performance of widely striped file reads.
    - Only 8 OSTs were available for these tests, so this was not tested.
- ORR/TRR may improve the performance of backward reads.
    - Again, few RPCs were generated for this test, so this was not tested.
- TRR on a multi-layered NRS policy environment can be simplified.
- ORR policy 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.

Test results summary

- The NRS framework with FIFO policy: no significant performance regressions.
  - Data and metadata operations tested at reasonably large scale.
- The CRR-N and ORR/TRR policies look promising for some use cases; CRR-N tends to smooth reads out, ORR/TRR improve performance for reads in some test cases.
  - May be useful in specific scenarios, or for more generic usage.
  - We may get the best of policies when they are combined in a multi-layer NRS arrangement.
- Further testing may be required.
  - CRR-N and ORR/TRR benefits at larger scale.
  - We ran out of large-scale resources for LUG, but will follow up on this presentation with more results.

Future tasks

- Decide whether the NRS framework with FIFO policy and perhaps others, can land soon.
  - Work out a test plan with the community if further testing is required.
- We should be able to perform testing at larger scale soon.
- Two policies operating at the same time should be useful.
- NRS policies as separate kernel modules?
- QoS policy.

xyratex

Advancing Digital Storage Innovation

Thank you!

Nikitas Angelinas
nikitas_angelinas@xyratex.com