Final Report for the SMP Node Affinity Sub-project of the Single Metadata Server Performance Improvements of the SFS-DEV-001 contract.

Revision History

Date	Revision	Author
10/02/12	Original	R. Henwood

Contents

Introduction

This document describes demonstration of sub-project 2.1 – SMP Node Affinity – within the OpenSFS Lustre Development contract SFS-DEV-001 signed 7/30/2011.

The SMP Node Affinity code is functionally complete. The purpose of this Milestone is to verify that the code performs acceptably in a production-like environment. In addition to achieving the Acceptance Criteria (recorded in the SMP Node Affinity Solution Architecture), SMP Node Affinity Performance will be measured as described below.

Performance Baseline

SMP Node Affinity code delivers a performance enhancement on multi-core MDSs. To demonstrate the completion of the SMP Node Affinity goals, a baseline performance is established to compare against. The baseline is chosen as Lustre 2.2.

SMP Node Affinity code arrived in Master as a series of 45 change-sets, containing over 14K LOC. These changes arrived over a six week period. Performance measurements are made with a Lustre build that includes the SMP Node Affinity patches as of July 1st 2012. The version of Lustre with SMP Node Affinity enabled is known in this document as Lustre 2.3.

Test Methodology

The following tools will be used to provide a production-like load. Performance of the MDS will be measured using Lnet selftest and mdtest.

Lnet selftest

In-order to generate a suitable load, high concurrency is required on the client side. High concurrency in this environment is measure as over one thousand RPCs/second from 16 clients.

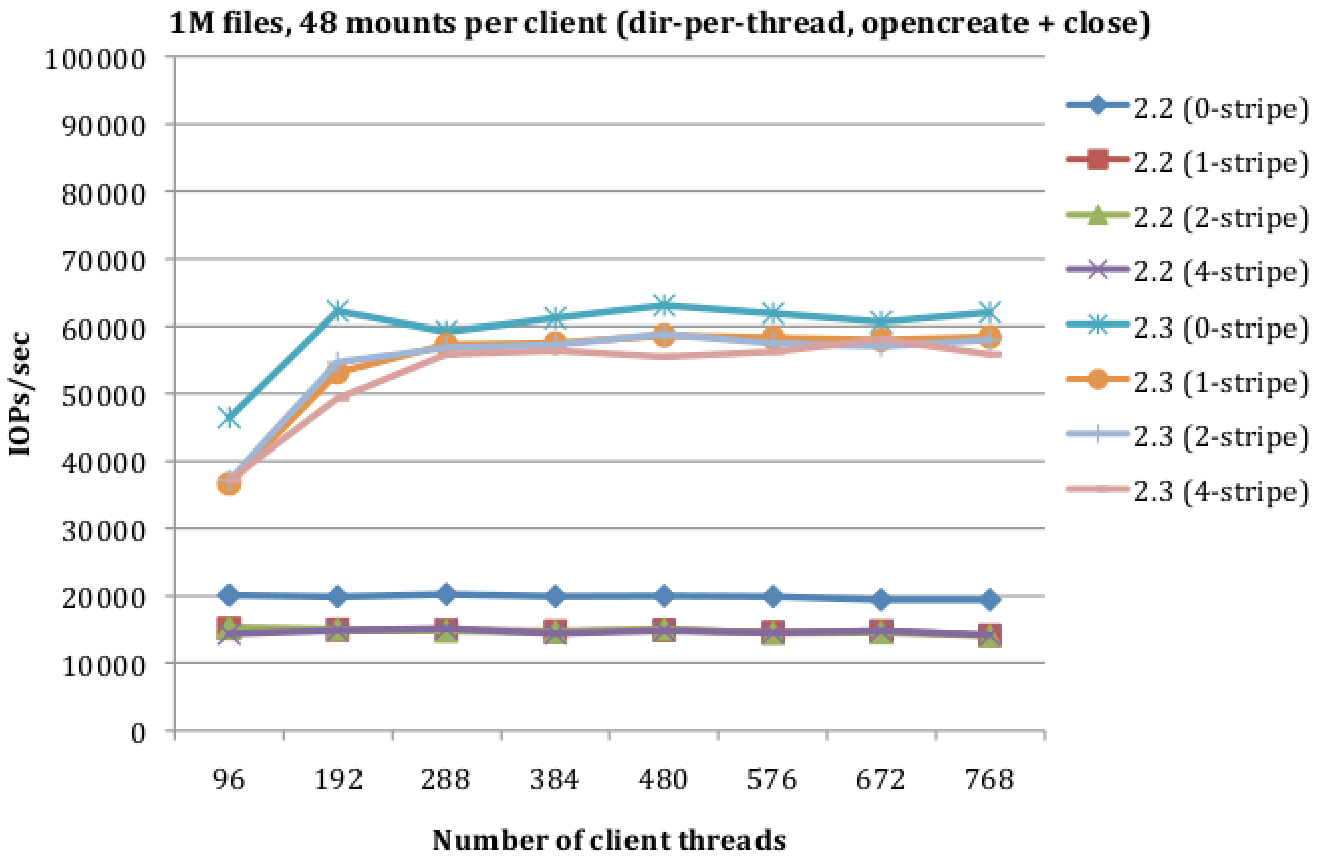
mdtest

Multiple mounts on each client

- Each thread operates on a individual mount.
- Sufficient workload for shared directory is achieved because the target directory has a separate inode for each mount on the client.

Summary of Demonstration

SMP Node Affinity patches display a significant speed-up in all meta-data operations.



The figure above illustrates the performance enhancement for one example case: create a new file with an increasing a number of clients.

Code Availability

Code with SMP Node Affinity patches applied is available as Lustre Master:

http://git.whamcloud.com/?p=fs/lustre-release.git;a=summary

Appendix 1: Detailed Demonstration Results

OPENSFS SMP NODE AFFINITY DEMONSTRATION

Liang Zhen (liang.zhen@intel.com)

Hardware

MDS node:

- 2 x Intel Xeon(R) X5650 2.67GHz Six-core Processor (2-HT each core), which will present as 24 CPUs on Linux
- 24GB DDR3 1333MHz Memory
- 2PT 40Gb/s 4X QSFP InfiniBand adapter card (Mellanox MT26428)
- 1 QDR IB port on motherboard
- SSD as external journal device (INTEL SSDSA2CW120G3), SATA II Enterprise Hard Drive as MDT (single disk, WDC WD2502ABYS-02B7A0)

OSS:

- 2 x AMD Opteron 6128 2.0GHz Eight-Core Processor
- 16GB DDR3 1333MHz Memory
- 2PT 40Gb/s 4X QSFP InfiniBand adapter card (Mellanox MT26428)
- 1 QDR IB port on motherboard
- 3 x 1TB SATA II Enterprise Hard Drive (single disk, WDC WD1003FBYX-01Y7B0)

Client:

- Quad-Core Intel E5507 2.26G/4MB/800
- Mellanox ConnectX 6 QDR Infiniband 40Gbps Controller (MTS3600Q-1BNC)
- 12GB DDR3 1333MHz E/R memory

Network:

- Infiniband between all the nodes: MTS3600Q managed QDR switch with 36 ports.

Test Methodology

The following tools will be used to provide a production-like load. Performance of the MDS will be measured using LNet selftest and mdtest.

1. LNet selftest

- In-order to generate a suitable load, high "concurrency" is required on the client side. High concurrency in this environment is measured as over one thousand RPCs from 16 clients.

2. mdtest

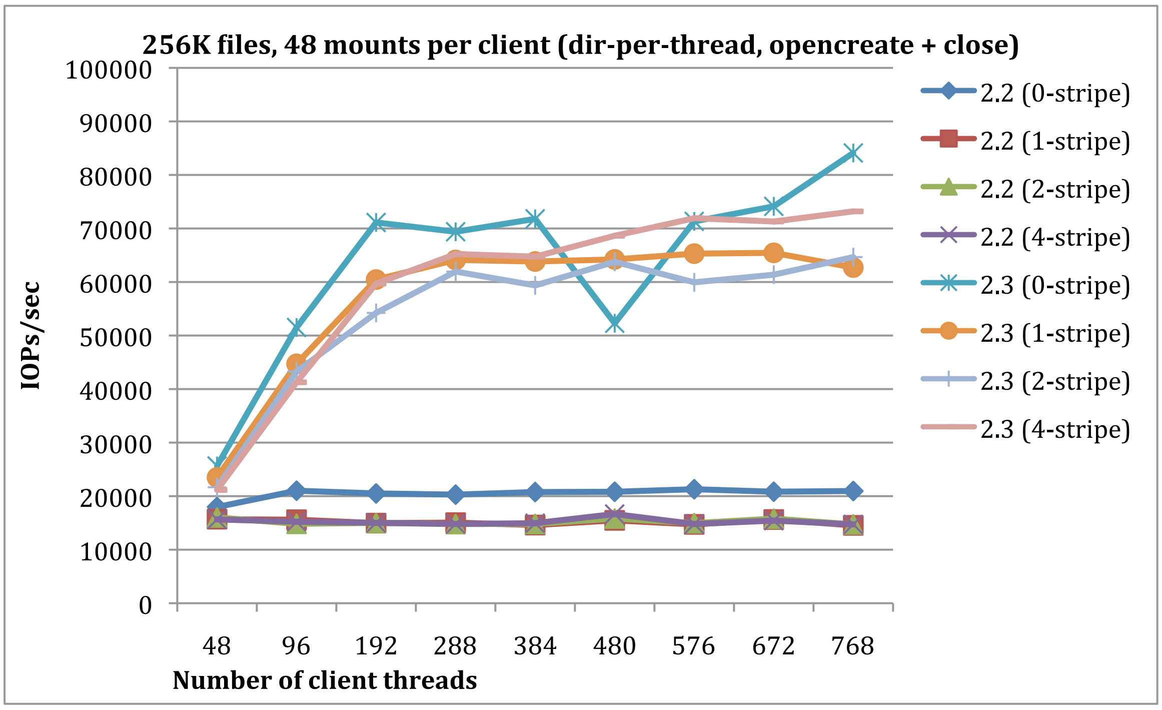
- Multiple mounts on each client.
    - Each thread works under a private mount.
    - Sufficient workload for "shared directory" is achievable because target directory has separate inode for each mount on client.

Test File System Configuration

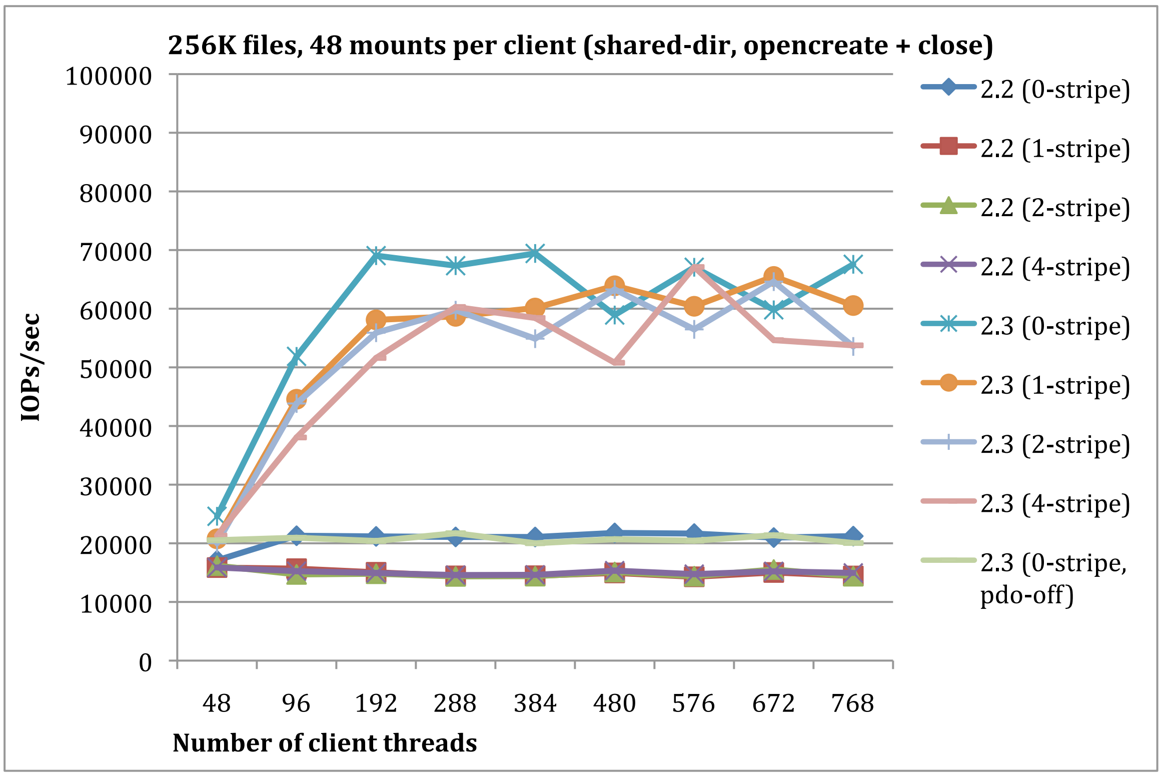
- 16 Clients, each with 48+ threads.
- 1 MDS, 3 OSS (6 OSTs on each OSS).
- Test repeated three times. Mean values are recorded.
- Test completed with both Lustre 2.2 and Lustre-Master.

Mdtest file creation performance (Total 256K files)

- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount and has a private working directory.
    - 2.3 creation performance is about 400%+ of 2.2

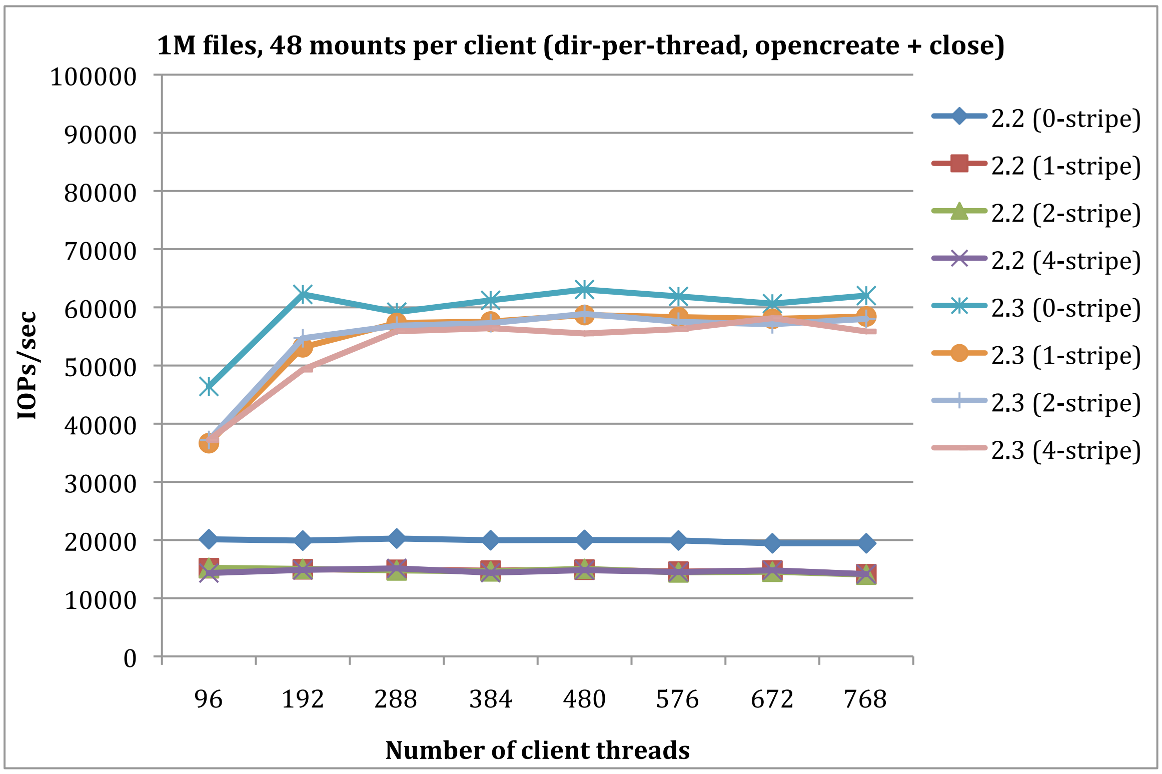


- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount, all threads share a target directory. 2.3 is also tested with turning off PDO.
    - 2.3 creation performance is about 350%+ of 2.2
    - Turning off PDO, shared-directory creation performance of 2.3 is similar to 2.2

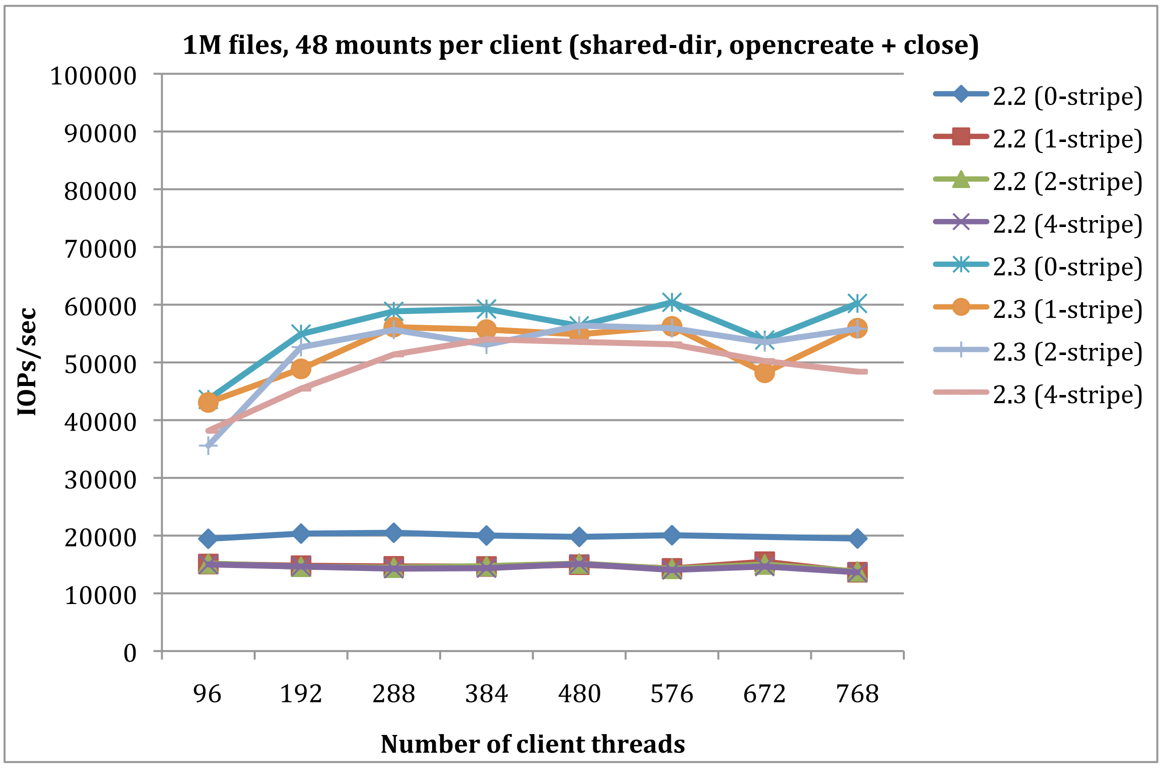


mdtest file creation performance (Total 1 million files)

- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount and has a private working directory.
  - 2.3 creation performance is about 370%+ of 2.2

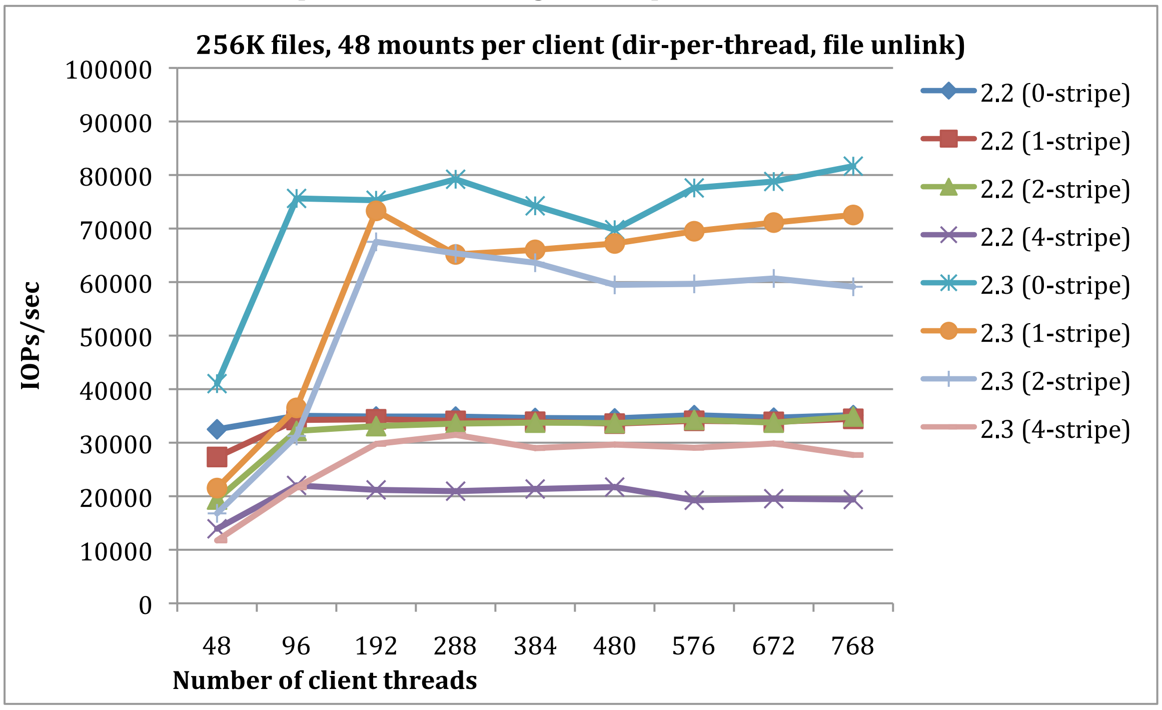


- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount, all threads share a target directory.
  - 2.3 creation performance is about 350%+ of 2.2

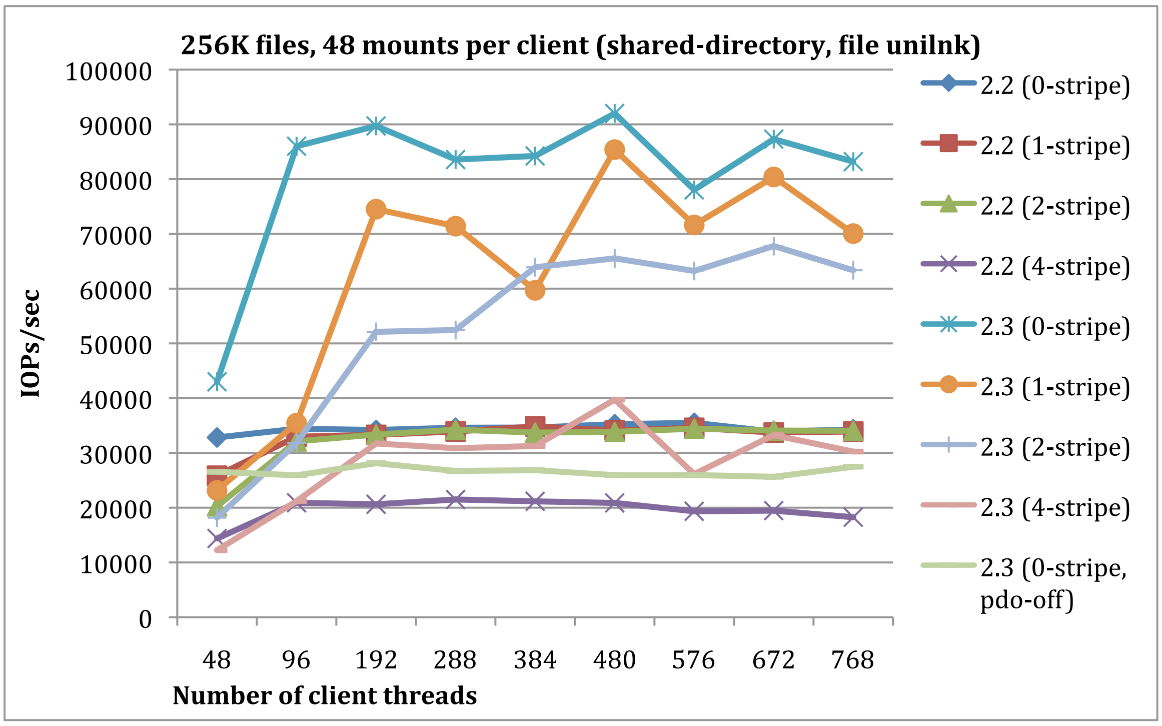


mdtest file removal performance (Total 256K files)

- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount and has a private working directory.
    - 2.3 file unlink performance is 250-300% of 2.2 (depends on stripecount)
    - Lustre client needs to send RPC for each OST object, that's the reason that performance will drop while increasing file stripecount

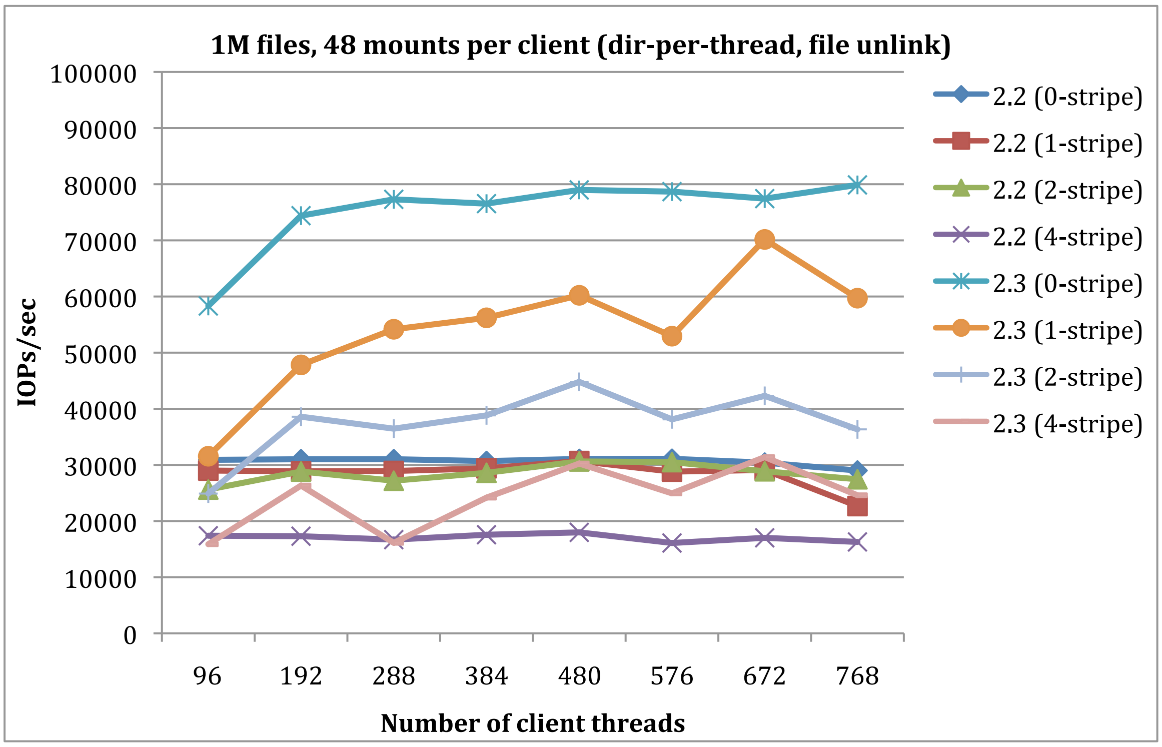


- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount, all threads share a target directory. 2.3 is also tested with turning off PDO.
    - 2.3 file unlink performance is 250-300% of 2.2 (depends on stripecount)
    - Lustre client needs to send RPC for each OST object, that's the reason that performance will drop while increasing file stripecount
    - Turning off PDO, shared-directory unlink performance of 2.3 is even lower than 2.2, it's probably because there're more contention on ldiskfs dir mutex in 2.3.

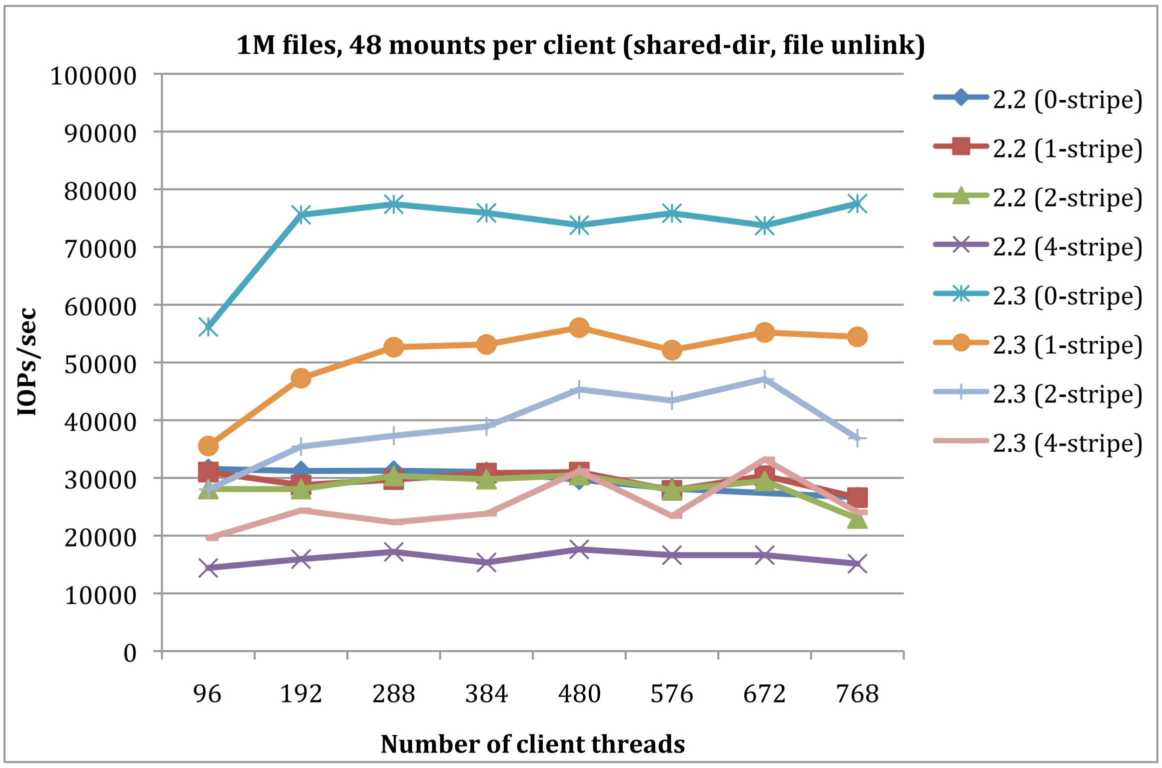


mdtest file removal performance (Total 1 million files)

- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount and has a private working directory.
  - 2.3 file unlink performance is 100-250% of 2.2 (depends on stripecount)
  - Lustre client needs to send RPC for each OST object, that's the reason that performance will drop while increasing file stripecount

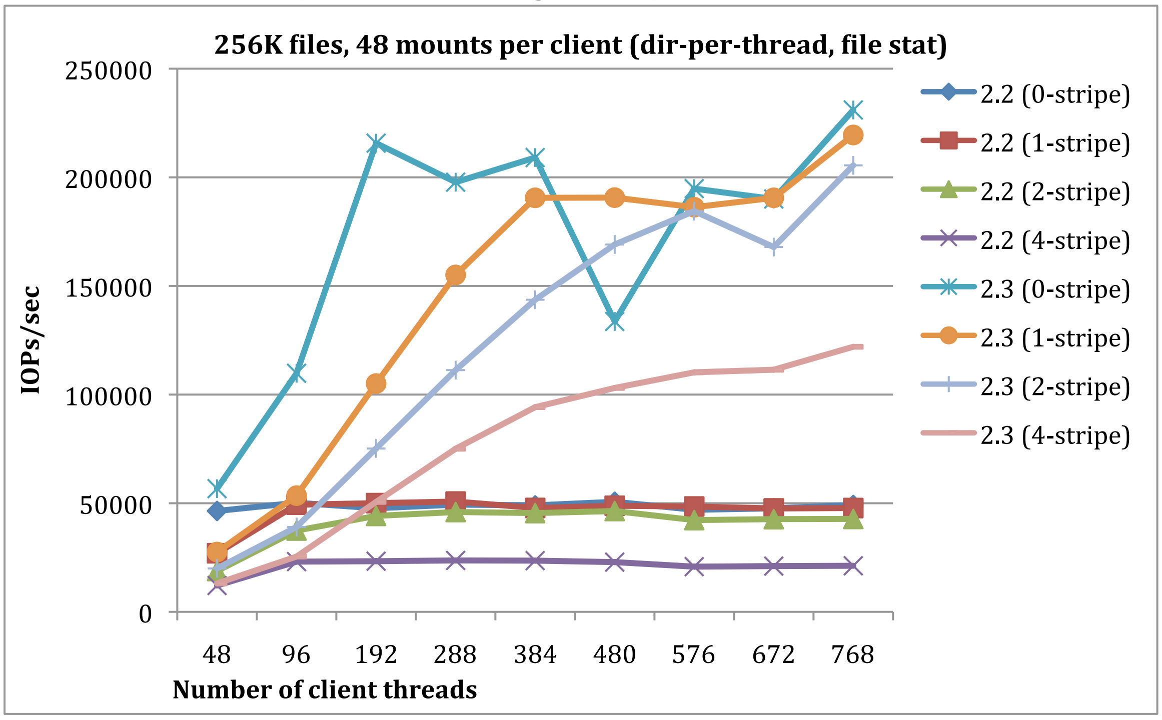


- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount, all threads share a target directory.
  - 2.3 file unlink performance is 100-250% of 2.2 (depends on stripecount)
  - Lustre client needs to send RPC for each OST object, that's the reason that performance will drop while increasing file stripecount

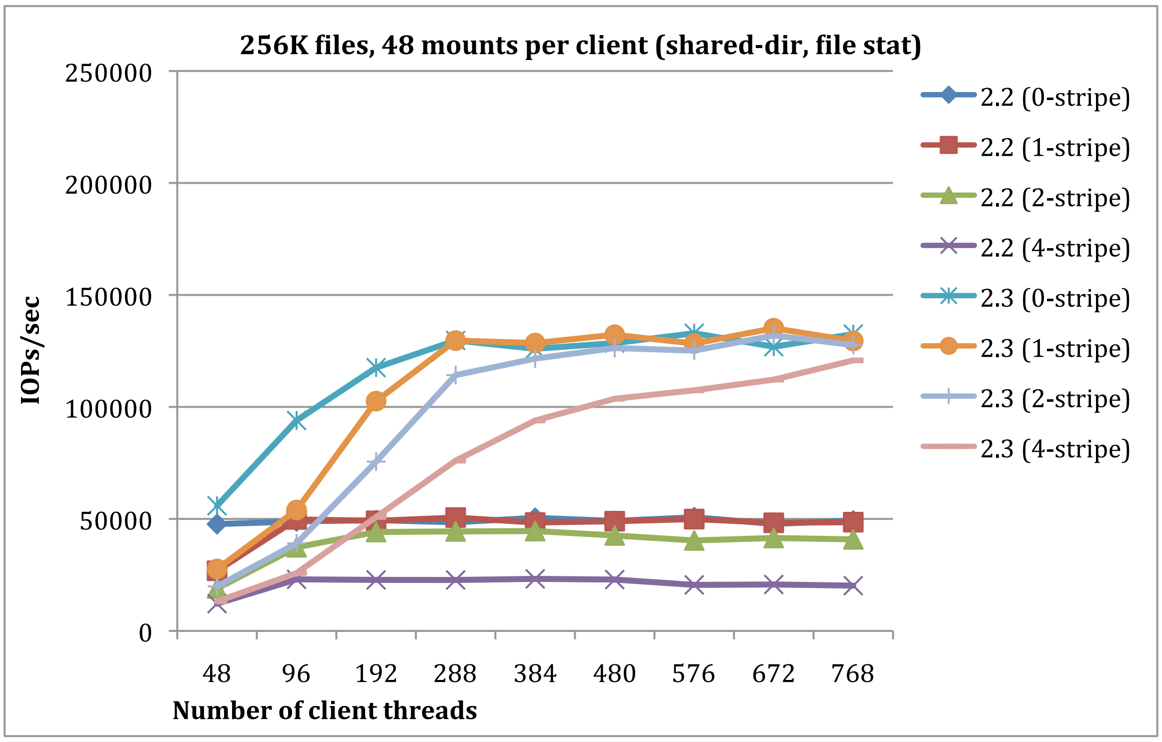


mdtest file stat performance (Total 256K files)

- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount and has a private working directory.
    - 2.3 file stat performance is 300-400% of 2.2 (depends on stripecount)
    - Lustre client needs to send RPC for each OST object, that's the reason that performance will drop while increasing file stripecount

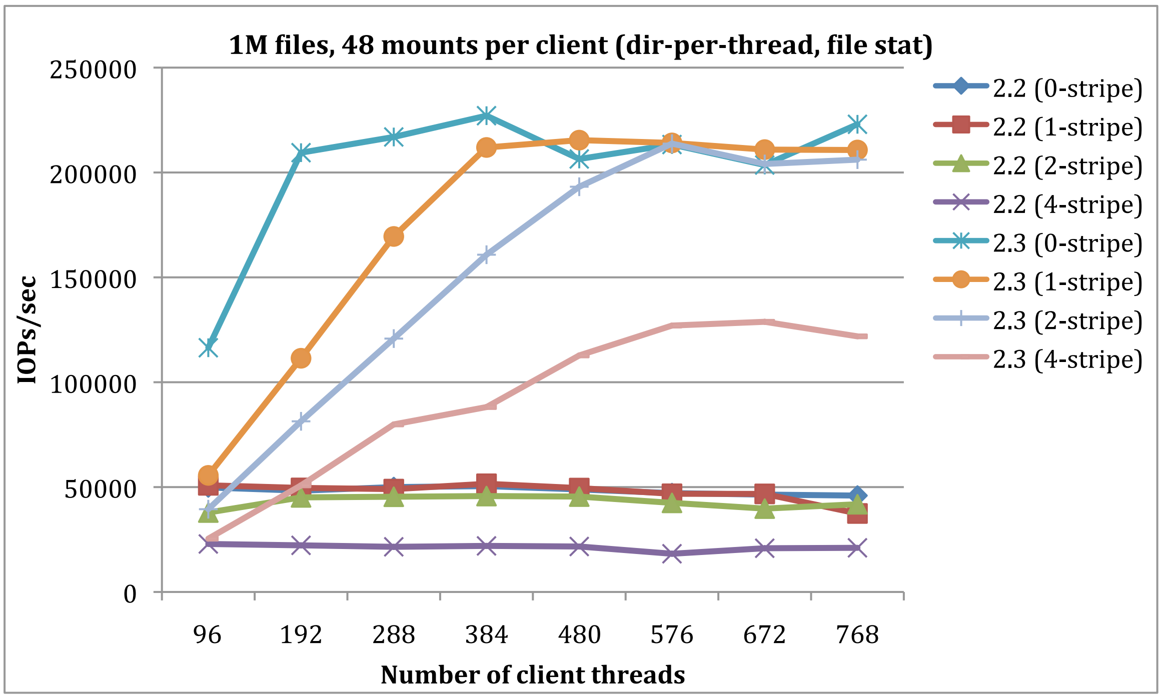


- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount, all threads share a target directory.
    - 2.3 file stat performance is 250% of 2.2 (depends on stripecount)
    - Lustre client needs to send RPC for each OST object, that's the reason that performance will drop while increasing file stripecount

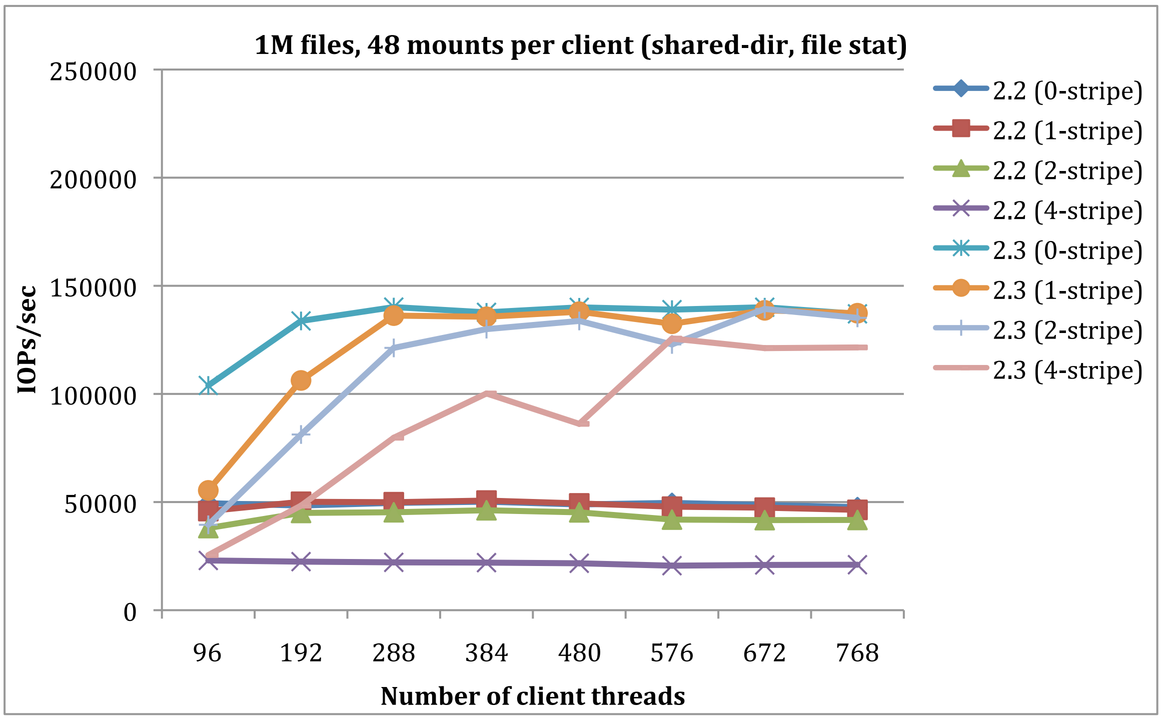


mdtest file stat performance (Total 1 million files)

- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount and has a private working directory.
    - 2.3 file stat performance is 400% of 2.2 (depends on stripecount)
    - Lustre client needs to send RPC for each OST object, that's the reason that performance will drop while increasing file stripecount

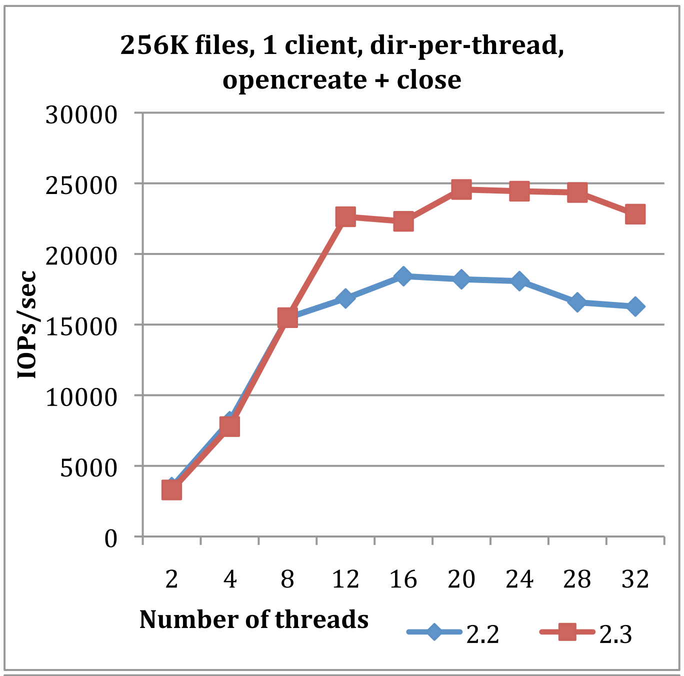


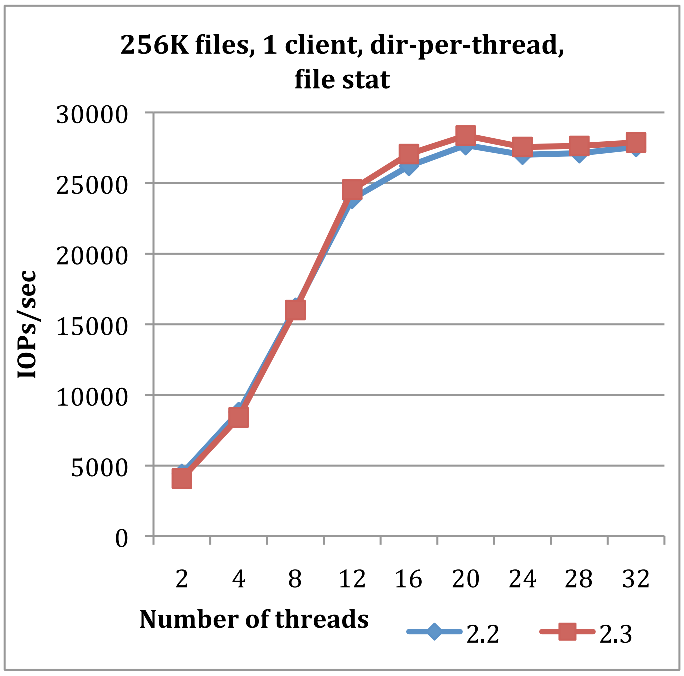
- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 48 threads, each thread is running under a private mount, all threads share a target directory.
    - 2.3 file stat performance is 250-300% of 2.2 (depends on stripecount)
    - Lustre client needs to send RPC for each OST object, that's the reason that performance will drop while increasing file stripecount

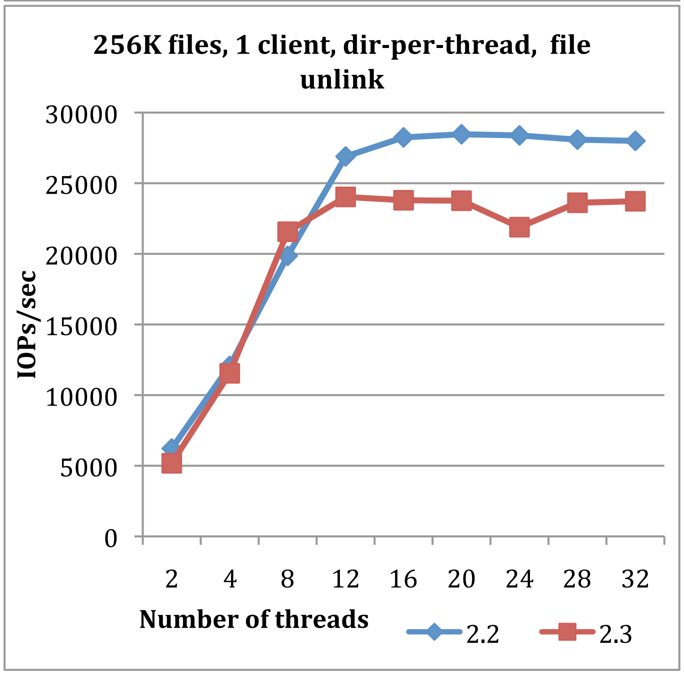


mdtest single client with multi-mount performance

- Single client, iterates over 2, 4, 8, 12, 16, 20, 24, 28, 32 threads. Each thread is running under a private mount and has a private working directory. File stripecount is always 1.
  - 2.3 file creation performance from a single client (multi-mount) is 30% better than 2.2
  - 2.3 file stat performance from a single client (multi-mount) is similar to 2.2
  - 2.3 file unlink performance from a single client (multi-mount) is 15% worse than 2.2, we don't know the reason yet.





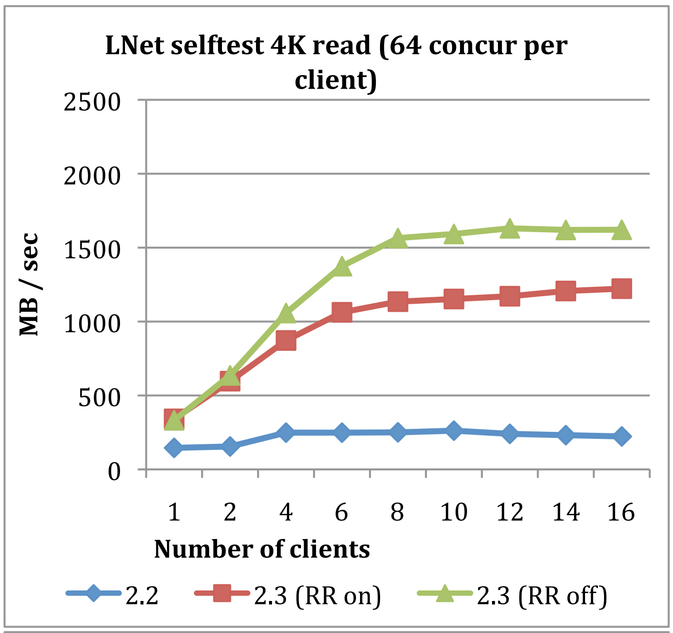


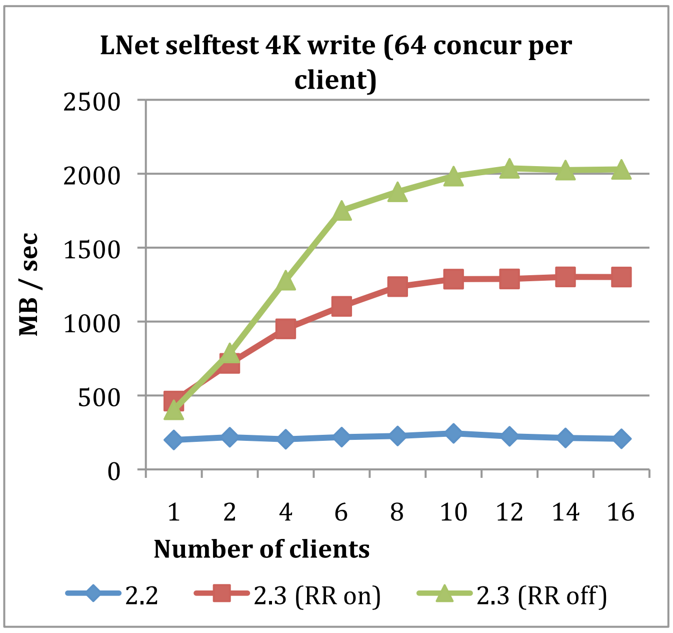
NB:

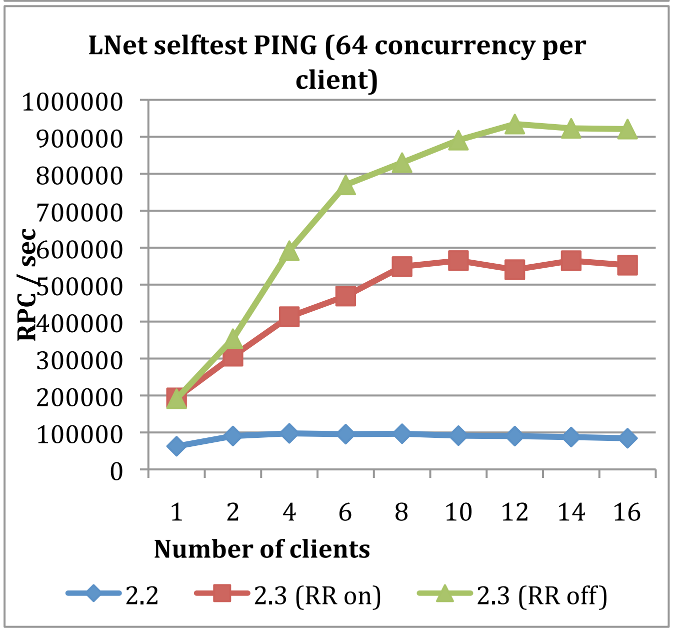
1. One thread case is not in the graph because patch mdtest (to support multi-mount) has a bug which failed one thread tests.

LNet Selftest performance

- Run “ping” and 4K brw read/write of LNet selftest
- Iterate over 1, 2, 4, 6, 8, 10, 12, 14, 16 clients, each client has 64 concurrencies (logic thread)
- Portal Round-Robin, it’s a new thing in 2.3 LNet
    - LNet peers are hashed to CPT (CPU partitions) by NID
    - Portal RR (Round-Robin) is OFF
      LND threads will fill incoming requests in buffers posted on local CPT, and deliver them to upper layer threads on local CPT (LNet selftest service threads, or ptlrpc service threads). If NID is not evenly distribute by hash(i.e: small site w/o enough clients), then it’s not good for CPU load balancing on MDS.
    - Portal RR (Round-Robin) is ON
      LND threads will round-robin use buffers on all CPTs to receive incoming requests, and deliver them to upper layer service threads that are running on CPT the buffer belonging.
- Portal Round-Robin, it’s a new thing in 2.3 LNet
    - 2.3 4K read & write performance is 600%-700% of 2.2 while turning off Portal RR
    - 2.3 4K read & write is 500% of 2.2 while turning on Portal RR
    - 2.3 ping is 900% of 2.2 with Portal RR-OFF, 600% of 2.2 with Portal RR-ON

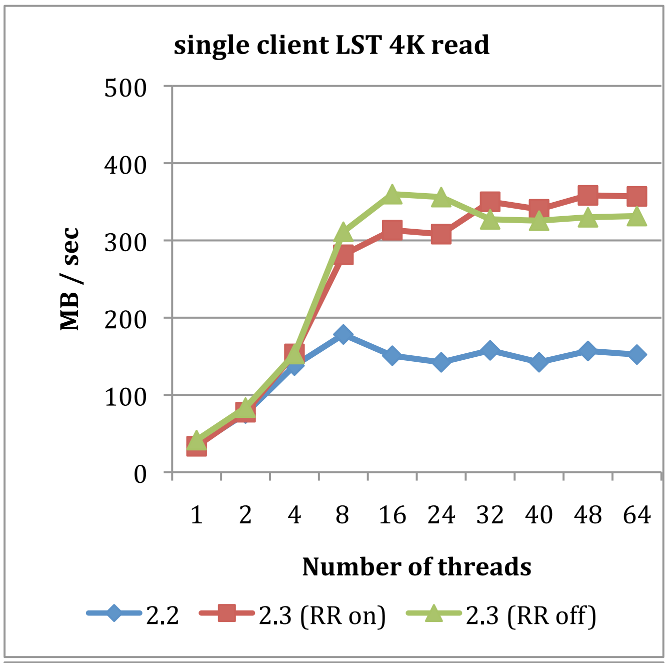


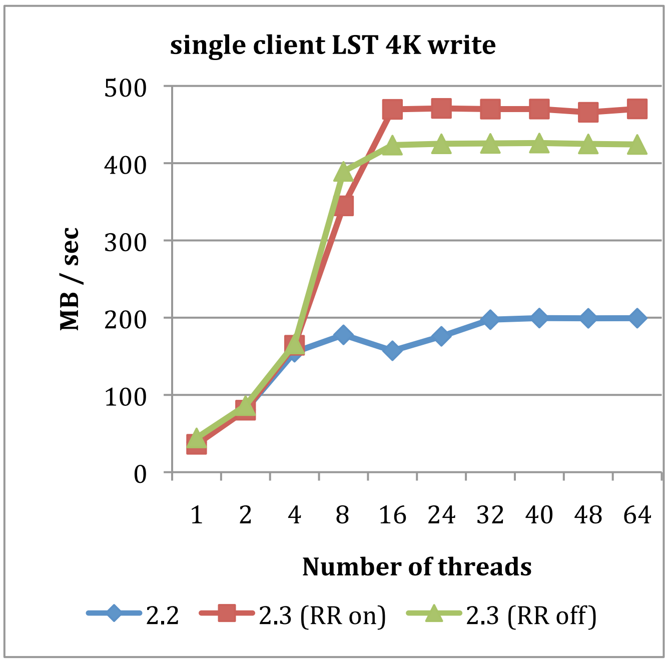


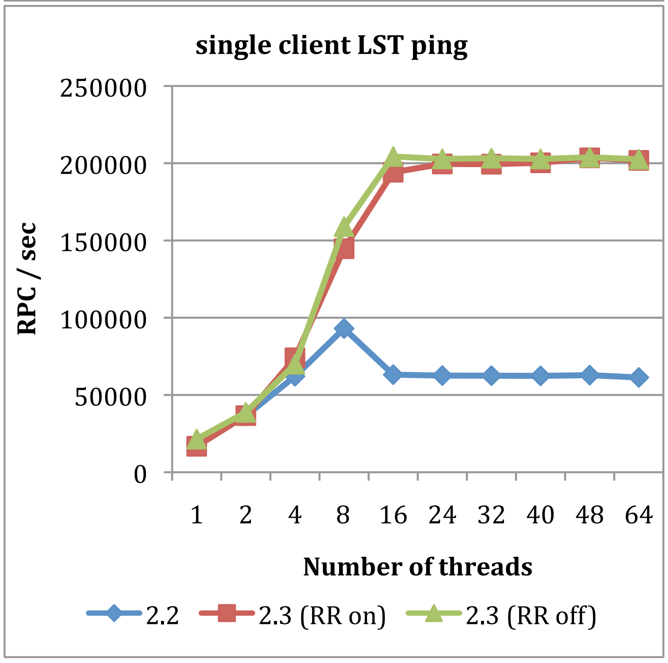


LNet selftest single client performance

- Run LNet selftest with a single client, iterate [1, 2, 4, 8, 16, 24, 32, 40, 48, 64] concurrency







Performance tests for different CPT configurations

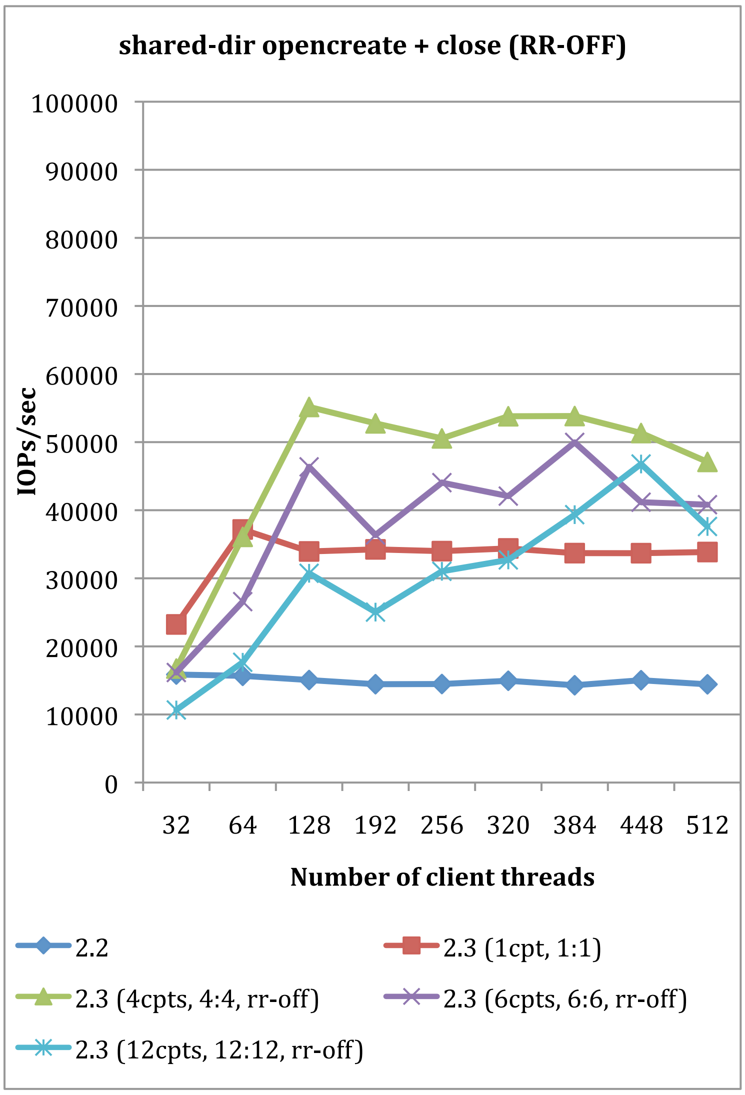
MDS has two six-core processors (2 HTs each core), which presents 24 CPUs in Linux. It's tested with these configurations:

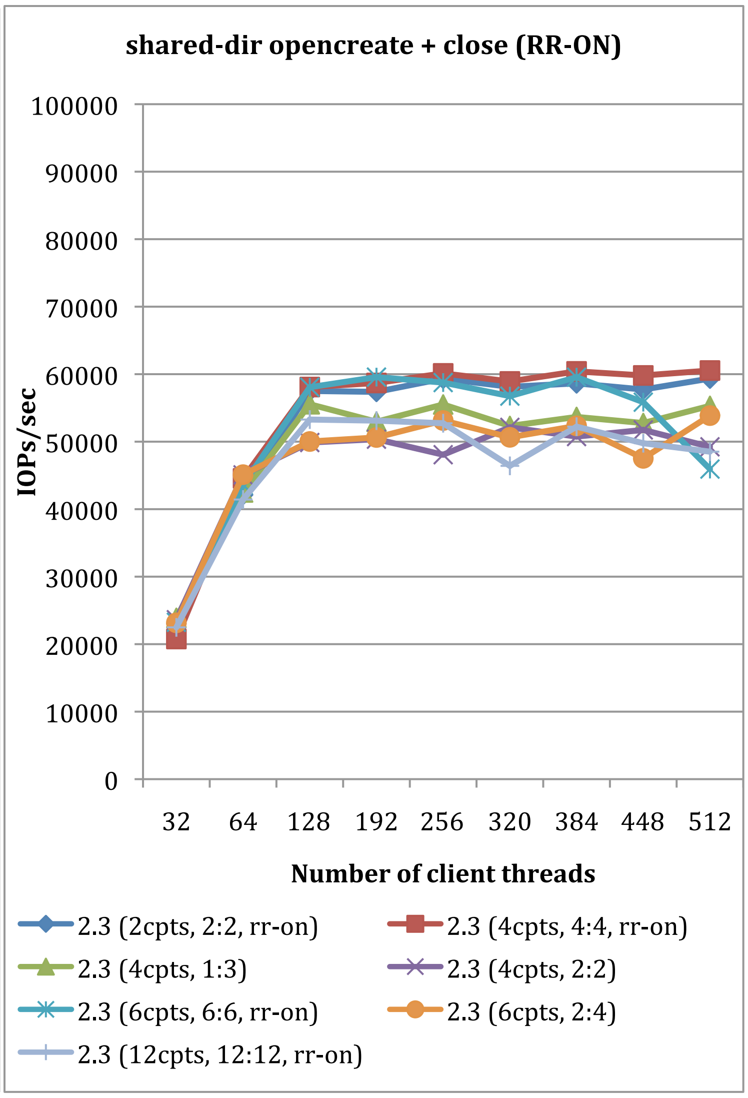
- Portal RR-OFF
  - 2.2 , It doesn't support Portal RR or CPT
  - 2.3 (1cpt, 1:1)
    One CPT which contains all CPUs, LNet (LND) and ptlrpc service are both running on this CPT
  - 2.3 (4cpts, 4:4, rr-off)
    4 CPTs, each CPT contains 3 cores (6 HTs), LNet (LND) and ptlrpc service are both running on all CPTs, LND threads always deliver requests to ptlrpc service threads on local CPT
  - 2.3 (6cpts, 6:6, rr-off)
    6 CPTs, each CPT contains 2 cores (4 HTs), LNet (LND) and ptlrpc service are both running on all CPTs, LND threads always deliver requests to ptlrpc service threads on local CPT
  - 2.3 (12cpts, 12:12, rr-off)
    12 CPTs, each CPT contains 1 cores (2 HTs), LNet (LND) and ptlrpc service are both running on all CPTs, LND threads always deliver requests to ptlrpc service threads on local CPT

- Portal RR-ON
  - 2.3 (2cpts, 2:2 rr-on)
    2 CPTs, each CPT contains 6 cores (12 HTs), LNet (LND) and ptlrpc service are both running on all CPTs, LND threads will round-robin deliver request to ptlrpc service threads on all CPTs
  - 2.3 (4cpts, 4:4 rr-on)
    4 CPTs, each CPT contains 3 cores (6 HTs), LNet (LND) and ptlrpc service are both running on all CPTs, LND threads will round-robin deliver request to ptlrpc service threads on all CPTs
  - 2.3 (4cpts, 1:3)
    4 CPTs, each CPT contains 3 cores (6 HTs), LNet (LND) is running on 1 CPT, ptlrpc service is running on the other 3 CPTs, LND threads will round-robin deliver request to ptlrpc service threads on those three CPTs
  - 2.3 (4cpts, 2:2)
    4 CPTs, each CPT contains 3 cores (6 HTs), LNet (LND) is running on 2 CPTs, ptlrpc service is running on the rest 2 CPTs, LND threads will round-robin deliver request to ptlrpc service threads on those 2 CPTs
  - 2.3 (6cpts, 6:6, rr-on)
    4 CPTs, each CPT contains 2 cores (4 HTs), LNet (LND) and ptlrpc service are both running on all CPTs, LND threads will round-robin deliver request to ptlrpc service threads on all CPTs
  - 2.3 (6cpts, 2:4, rr-on)
    6 CPTs, each CPT contains 2 cores (4 HTs), LNet (LND) is running on 2 CPTs, ptlrpc service is running on the other 4 CPTs, LND threads will round-robin deliver request to ptlrpc service threads on those 4 CPTs
  - 2.3 (12cpts, 12:12, rr-on)
    12 CPTs, each CPT contains 1 cores (2HTs), LNet (LND) and ptlrpc service are both running on all CPTs, LND threads will round-robin deliver request to ptlrpc service threads on all CPTs

Mdtest share-directory file creation performance for different CPT configurations

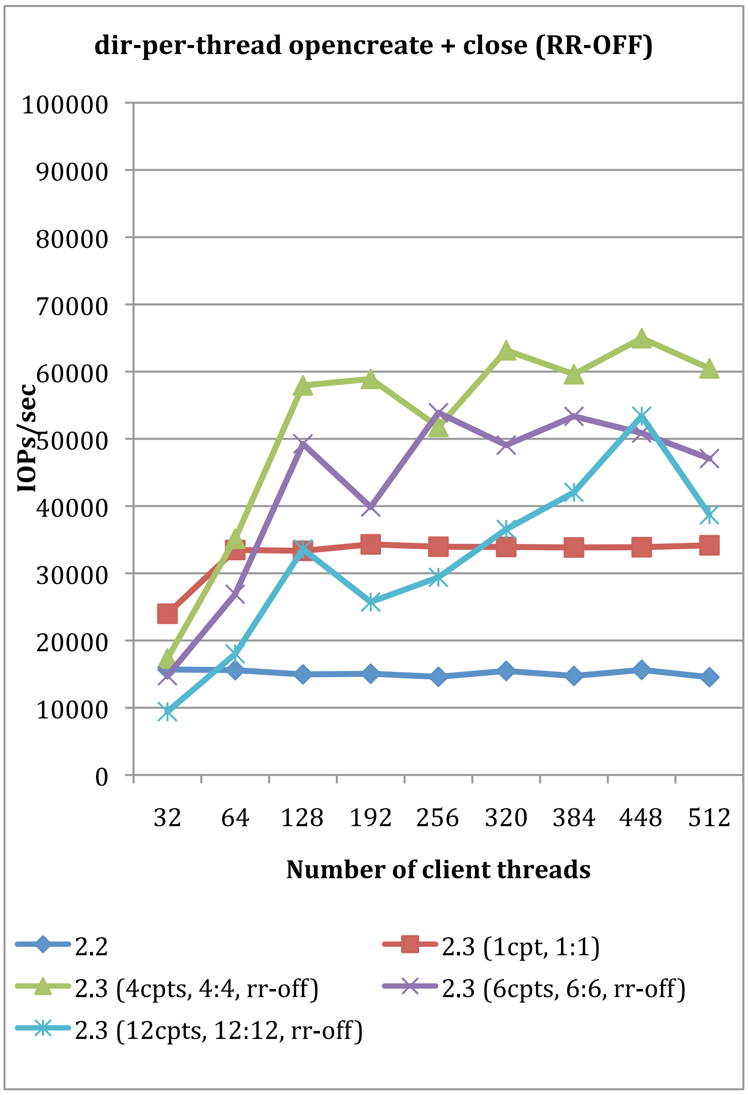
- Total 256K files
- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 32 threads, each thread is running under a private mount, all threads share a target directory
- 4 CPTs gives the best performance, which is the default value calculated from CPU numbers on server
- Turning on Portal RR gives a little better performance
- 2.3 generally has much better performance than 2.2, even configured to a single CPT, 2.3 performance is still much better than 2.2
- Performance of 2.3 with single CPT is worse than 2.3 with multiple CPTs.

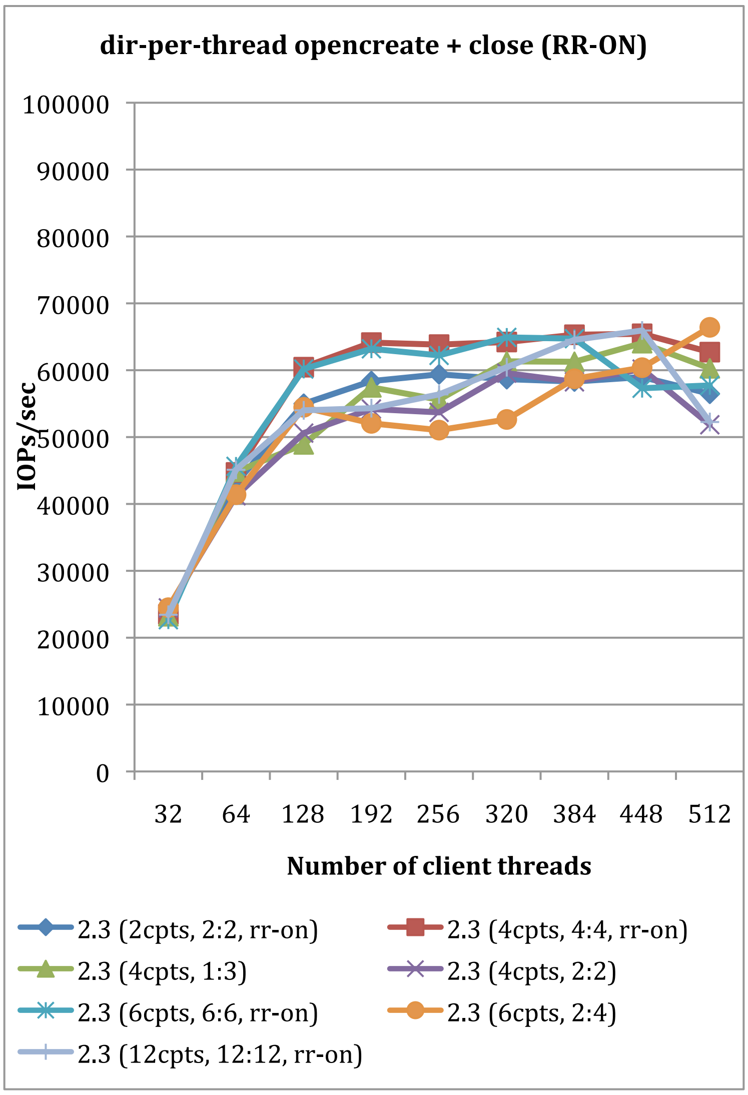




Mdtest directory-per-thread file creation performance for different CPT configurations

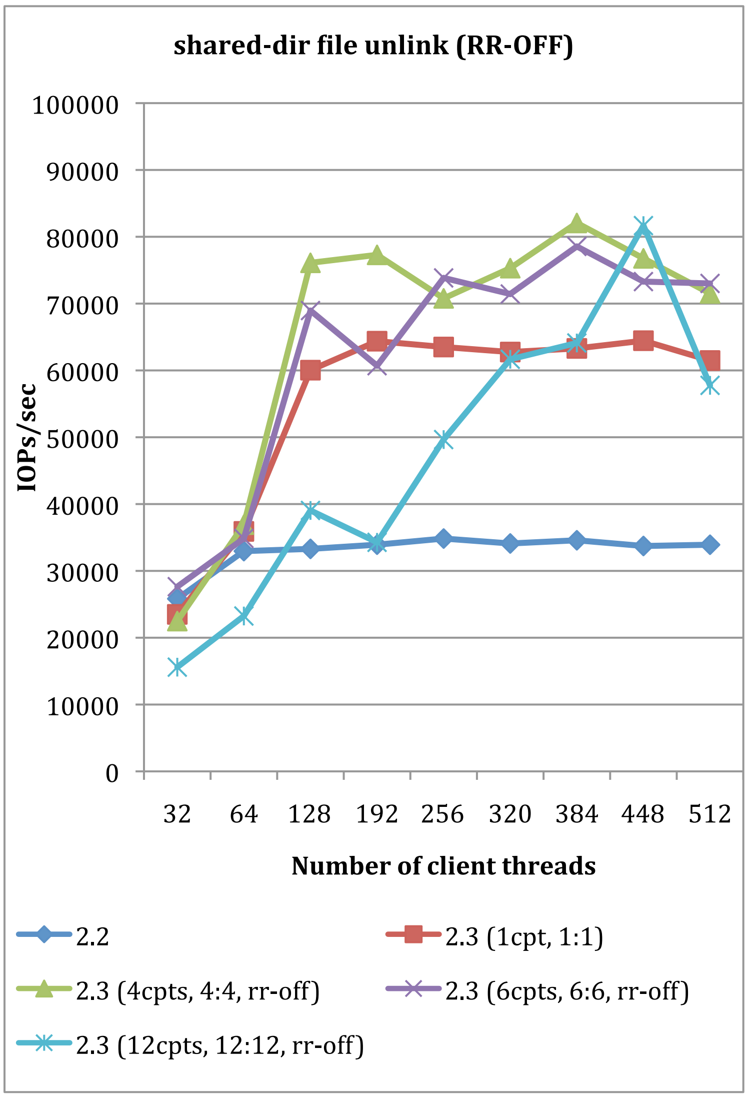
- Total 256K files
- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 32 threads, each thread is running under a private mount and has a private working directory.
- 4 CPTs gives the best performance, which is the default value calculated from CPU numbers on server
- Turning on Portal RR gives a little better performance
- 2.3 generally has much better performance than 2.2, even configured to a single CPT, 2.3 performance is still much better than 2.2
- Performance of 2.3 with single CPT is worse than 2.3 with multiple CPTs.

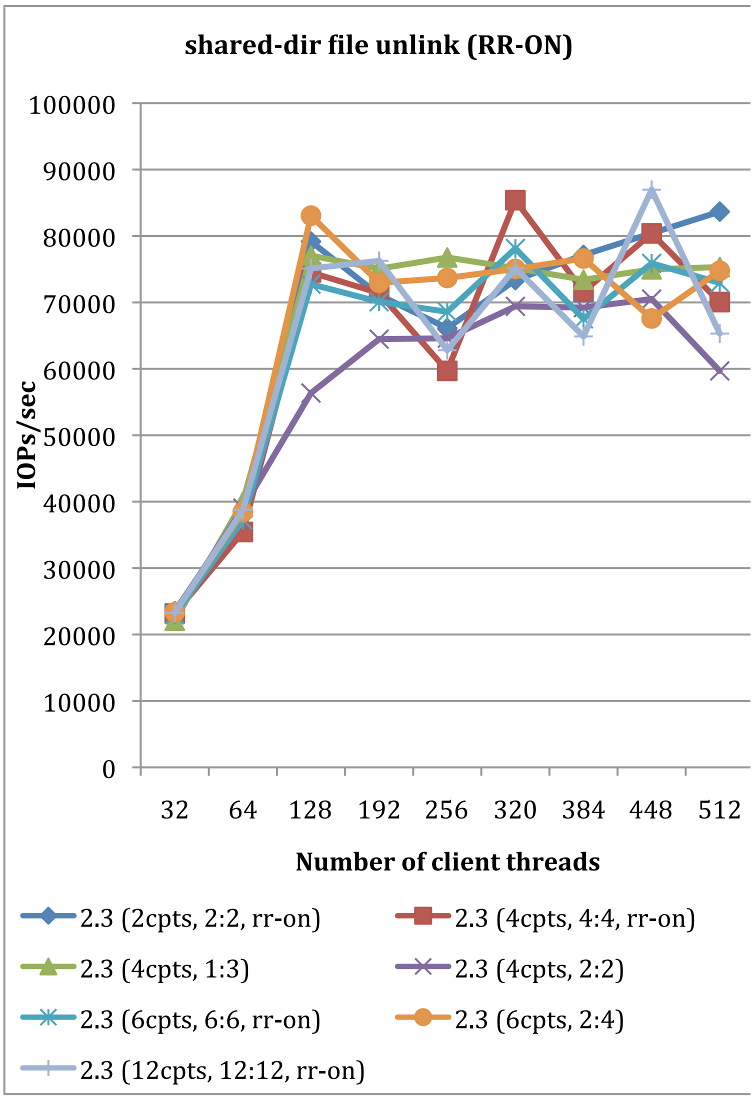




Mdtest shared-directory file removal performance for different CPT configurations

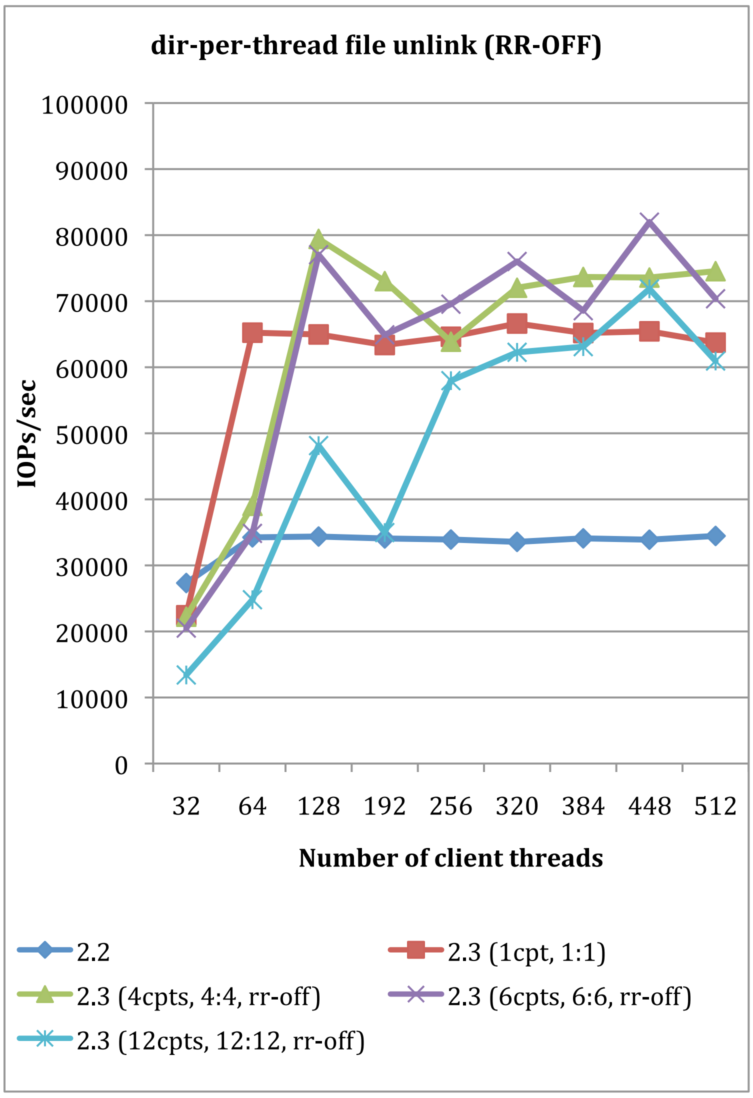
- Total 256K files
- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 32 threads, each thread is running under a private mount, all threads share a target directory
- 4 CPTs gives the best performance, which is the default value calculated from CPU numbers on server
- Turning on/off Portal RR give similar performance
- 2.3 generally has much better performance than 2.2, even configured to a single CPT, 2.3 performance is still much better than 2.2
- Performance of 2.3 with single CPT is worse than 2.3 with multiple CPTs.

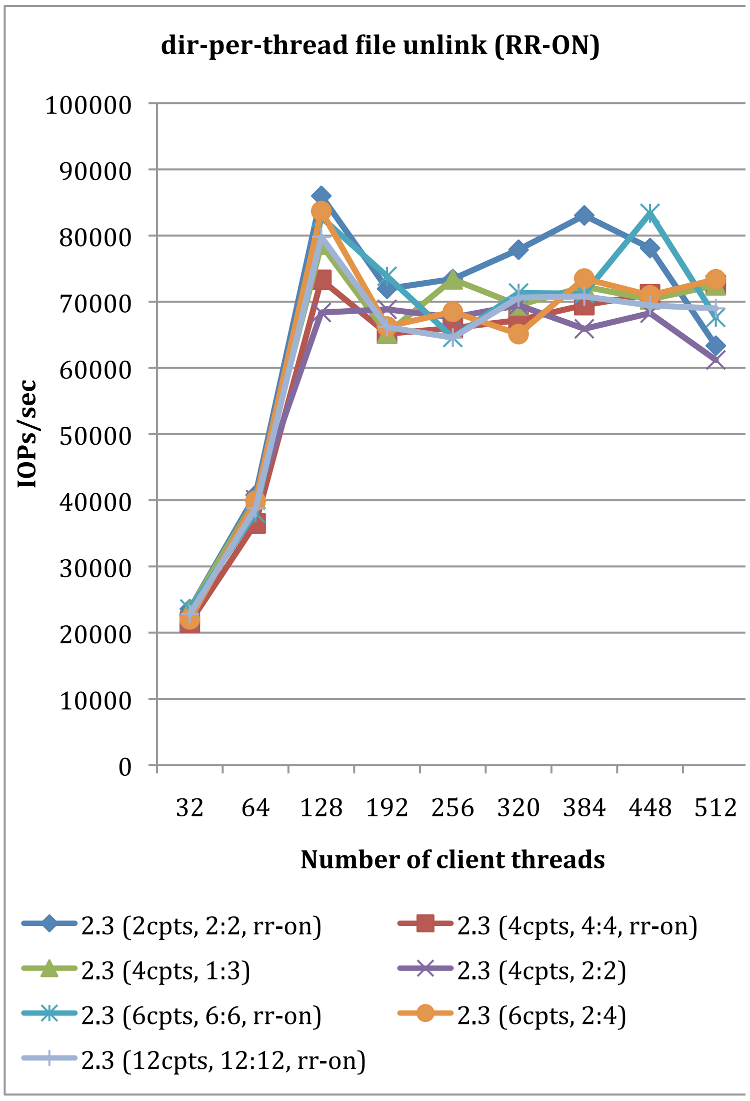




Mdtest directory-per-thread file removal performance for different CPT configurations

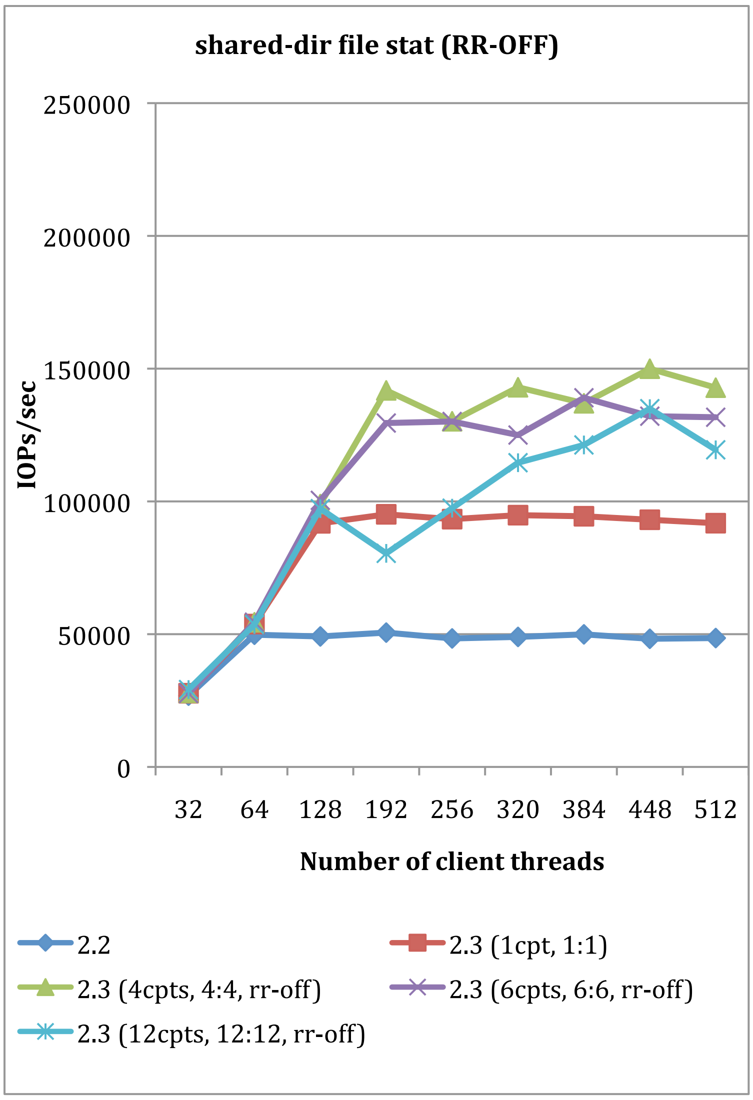
- Total 256K files
- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 32 threads, each thread is running under a private mount and has a private working directory.
- 4 CPTs gives the best performance, which is the default value calculated from CPU numbers on server
- Turning on/off Portal RR give similar performance
- 2.3 generally has much better performance than 2.2, even configured to a single CPT, 2.3 performance is still much better than 2.2

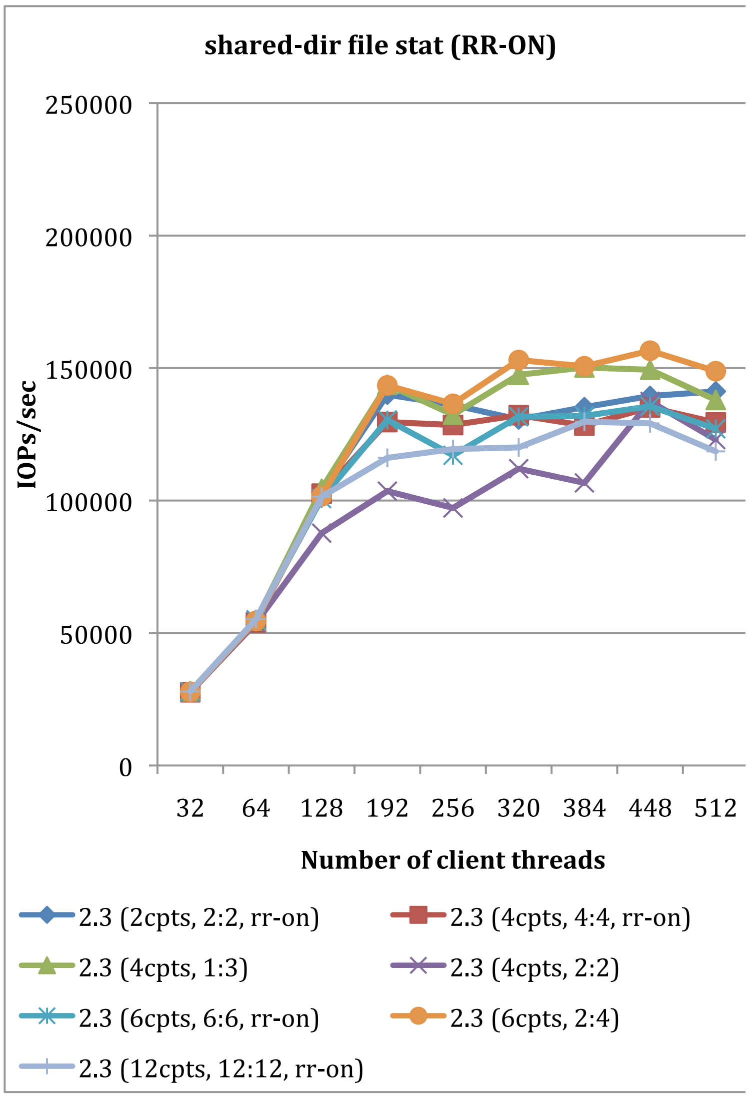




Mdtest shared-directory file stat performance for different CPT configurations

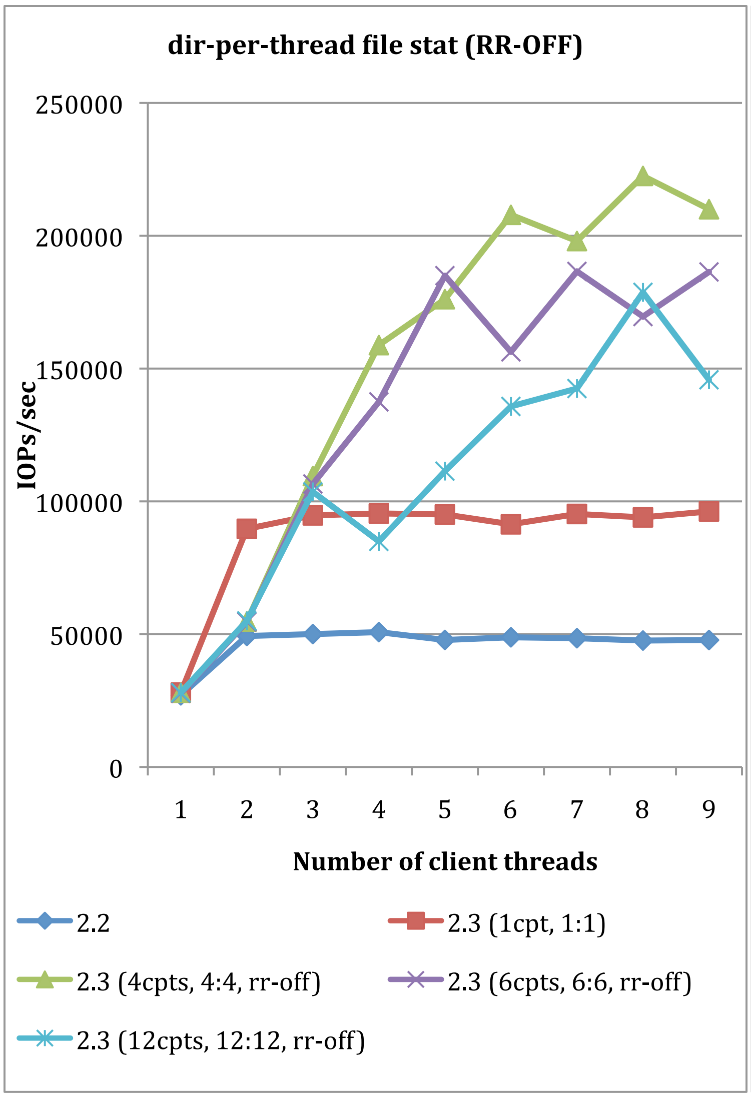
- Total 256K files
- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 32 threads, each thread is running under a private mount, all threads share a target directory
- 4 CPTs gives the best performance, which is the default value calculated from CPU numbers on server
- Turning on/off Portal RR give similar performance
- 2.3 generally has much better performance than 2.2, even configured to a single CPT, 2.3 performance is still much better than 2.2
- Performance of 2.3 with single CPT is worse than 2.3 with multiple CPTs.

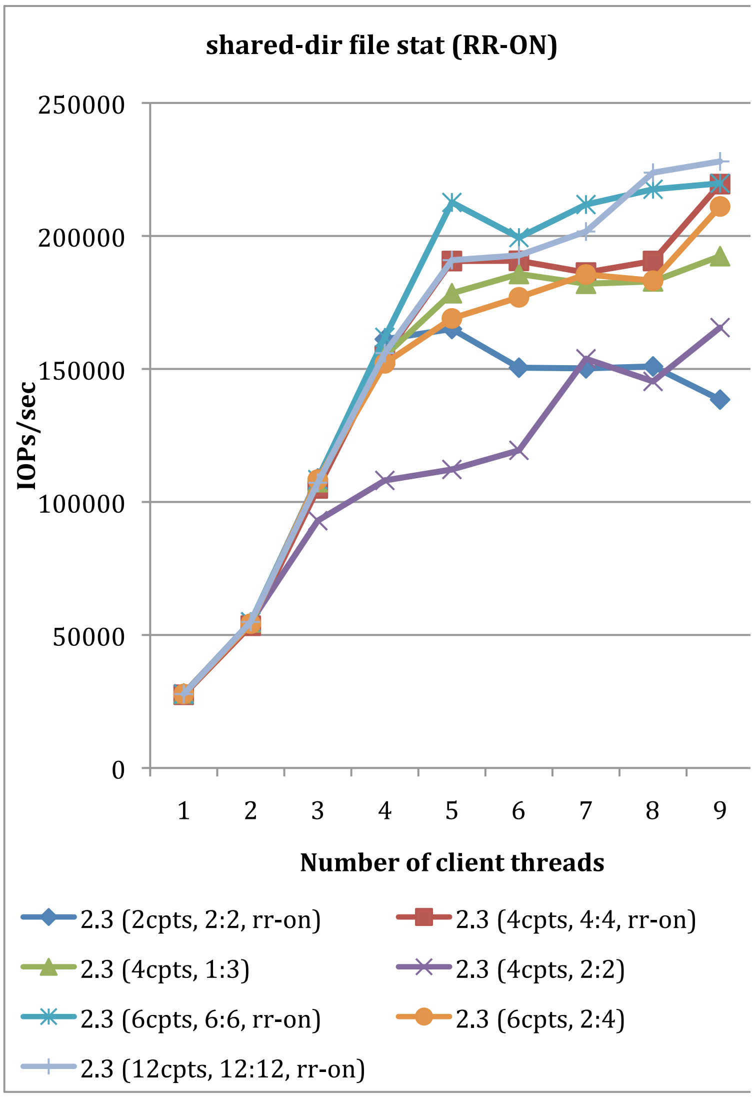




Mdtest directory-per-thread file stat performance for different CPT configurations

- Total 256K files
- Iterate over 1, 2, 4, 6, 8, 12, 16 clients, each client has 32 threads, each thread is running under a private mount and has a private working directory.
- 4 CPTs gives the best performance, which is the default value calculated from CPU numbers on server
- Turning on/off Portal RR give similar performance
- 2.3 generally has much better performance than 2.2, even configured to a single CPT, 2.3 performance is still much better than 2.2
- Performance of 2.3 with single CPT is worse than 2.3 with multiple CPTs.


dir-per-thread file stat (RR-OFF)
IOPs/sec
Number of client threads
2.3 (1cpt, 1:1)
2.3 (4cpts, 4:4, rr-off)
2.3 (6cpts, 6:6, rr-off)
2.3 (12cpts, 12:12, rr-off)


shared-dir file stat (RR-ON)
IOPs/sec
Number of client threads
2.3 (2cpts, 2:2, rr-on)
2.3 (4cpts, 4:4, rr-on)
2.3 (4cpts, 1:3)
2.3 (4cpts, 2:2)
2.3 (6cpts, 6:6, rr-on)
2.3 (6cpts, 2:4)
2.3 (12cpts, 12:12, rr-on)