Demonstration Milestone Completion for the LFSCK 3 (MDT-MDT Consistency) Sub-project 3.2 on the Lustre* File System FSCK Project of the SFS-DEV-001 contract.

Revision History

Date	Revision	Author
15/03/10	Original	R. Henwood

Table of Contents

Introduction

The following milestone completion document applies to Subproject 3.3 – LFSCK 3: MDT-MDT Consistency. This project is recorded in Amendment No. 1 on the OpenSFS Lustre Development contract SFS-DEV-001 agreed October 10, 2012.
The LFSCK 3: MDT-MDT Consistency code is functionally complete and recorded in the Implementation Milestone. The purpose of this Milestone is to verify the code performs acceptably in a production-like environment. In addition to completing all the Test Scenarios (demonstrated for the Implementation Milestone,) LFSCK 3: MDT-MDT Consistency Performance has been measured as recorded below.
All tests were executed on the OpenSFS Functional Test Cluster. Details of the hardware are available in Appendix A. For all the tests, Lustre software Master with LFSCK 3 patches was used.

Correctness Test Coverage

1. `sanity-lfsck.sh`
   Test will be executed within Autotest and results automatically recorded in Maloo. Test will be automatically completed, triggered by a Gerrit check-in with commit message "`Test-Parameters: envdefinitions=ENABLE_QUOTA=yes mdtcount=2 testlist=sanity-lfsck`". All test cases must pass.
2. `sanity-scrub.sh`
   Test will be executed within Autotest and results automatically recorded in Maloo. Test will be automatically completed, triggered by a Gerrit check-in with commit message "`Test-Parameters: envdefinitions=ENABLE_QUOTA=yes testlist=sanity-scrub`". All test cases must pass.
3. Standard review tests
   The standard collection of review tests (currently including sanity, sanityn, replay-single, conf-sanity, recovery-small, replay-ost-single, insanity, sanity-quota, sanity-sec, lustre-rsync-test, lnet-selftest, and mmp) will be executed within Autotest and results automatically recorded in Maloo. Tests will be automatically completed, triggered by a Gerrit check-in. All test cases should pass except for some known test failures unrelated to the LFSCK functionality.

The sanity-lfsck.sh and sanity-scrub.sh scripts are standard review tests that will be run and recorded. All previous LFSCK functionality is also tested with these scripts. For the specific case of MDT-MDT consistency, the test cases are:

- test 2e: The namespace LFSCK can detect inconsistent remote MDT-object's linkEA and repair it.
- test 6a: LFSCK can resume from the last checkpoint which is at the first-stage scanning.
- test 6b: LFSCK can resume from the last checkpoint which is at the second-stage scanning.
- test 7a: Non-stopped LFSCK can auto resume (the first-stage scanning) after the MDS restart.
- test 7b: Non-stopped LFSCK can auto resume (the second-stage scanning) after the MDS restart.
- test 9a: LFSCK speed is controllable during the first-stage scanning.
- test 9b: LFSCK speed is controllable during the second-stage scanning.
- test 10: During the LFSCK check/repair system inconsistency, the client can still access the

system normally.

- test 22a: Repair unmatched name entry and MDT-object pairs (1). The parent_A references the child directory via some name entry, but the child directory back references another parent_B via its ".." name entry. The parent_B does not exist. Then the namespace LFSCK will repair the child directory's ".." name entry to reference the parent_A.
- test 22b: Repair unmatched name entry and MDT-object pairs (2). The parent_A references the child directory via the name entry_B, but the child directory back references another parent_C via its ".." name entry. The parent_C exists, but there is no the name entry_B under the parent_C. Then the namespace LFSCK will repair the child directory's ".." name entry and its linkEA to reference the parent_A.
- test 23a: Repair dangling name entry (1). The name entry is there, but the MDT-object for such name entry does not exist. The namespace LFSCK should find out and repair the inconsistency as required.
- test 23b: Repair dangling name entry (2). The object_A has multiple hard links, one of them corresponding to the name entry_B. But there is something wrong for the name entry_B and cause entry_B to references non-exist object_C. During the first-stage scanning, the LFSCK will think the entry_B as dangling, and re-create the lost object_C. When the LFSCK comes to the second-stage scanning, it will find that the former re-creating object_C is not proper, and will try to replace the object_C with the real object_A.
- test 23c: Repair dangling name entry (3). The object_A has multiple hard links, one of them corresponding to the name entry_B. But there is something wrong for the name entry_B and cause entry_B to references non-exist object_C. During the first-stage scanning, the LFSCK will think the entry_B as dangling, and re-create the lost object_C. And then others modified the re-created object_C. When the LFSCK comes to the second-stage scanning, it will find that the former re-creating object_C maybe wrong and try to replace the object_C with the real object_A. But because object_C has been modified, so the LFSCK should NOT replace it to keep the data.
- test 24: Repair multiple-referenced name entry. Two MDT-objects back reference the same name entry via each own linkEA entry, but the name entry only references one MDT-object. The namespace LFSCK will remove the linkEA entry for the MDT-object that is not recognised. If such MDT-object has no other linkEA entry after the removing, then the LFSCK will add it as orphan under the .lustre/lost+found/MDTxxxx/.
- test 25: Repair invalid file type. The file type in the name entry does not match the file type claimed by the referenced object. The LFSCK will update the file type in the name entry.

- test 26a: Repair orphan MDT-object (1). The local name entry (back referenced by the MDT-object) is lost. The namespace LFSCK will add the missing local name entry back to the normal namespace.
- test 26b: Repair orphan MDT-object (2). The remote name entry (back referenced by the MDT-object) is lost. The namespace LFSCK will add the missing remote name entry back to the normal namespace.
- test 27a: Recreate the lost parent directory (1). The local parent (referenced by the MDT-object linkEA) is lost. The namespace LFSCK will re-create the lost parent as orphan.
- test 27b: Recreate the lost parent directory (2). The remote parent (referenced by the MDT-object linkEA) is lost. The namespace LFSCK will re-create the lost parent as orphan.
- test 29a: Repair invalid nlink count (1). The object's nlink attribute is larger than the object's known name entries count. The LFSCK will repair the object's nlink attribute to match the known name entries count.
- test 29b: Repair invalid nlink count (2). The object's nlink attribute is smaller than the object's known name entries count. The LFSCK will repair the object's nlink attribute to match the known name entries count.
- test 29c: Repair invalid nlink count (3). There are too many hard links to the object, and exceeds the object's linkEA limitation, as to NOT all the known name entries will be recorded in the linkEA. Under such case, the LFSCK should skip the nlink verification for this object.
- test 30: Recover the orphans from backend /lost+found. The namespace LFSCK will move the orphans from backend /lost+found directory (that is only valid for ldiskfs based backend) to normal client visible namespace or to the global visible ./lustre/lost+found/MDTxxxx/ directory.
- test 31a: Repair invalid name hash for striped directory (1). For the name entry under a striped directory, if the name hash does not match the shard (the case that some name entry should be inserted into other non-first shard, but inserted into the first shard by wrong), then the LFSCK will repair the bad name entry.
- test 31b: Repair invalid name hash for striped directory (2). For the name entry under a striped directory, if the name hash does not match the shard (the case that some name entry should be inserted into other non-second shard, but inserted into the second shard by wrong), then the LFSCK will repair the bad name entry.
- test 31c: Re-generate the lost master LMV EA for striped directory. For some reason, the master MDT-object of the striped directory may lost its master LMV EA. If nobody created files under the master directly after the master LMV EA lost, then the LFSCK should re-generate the master LMV EA.

- test 31d: Set broken striped directory (modified after broken) as read-only. For some reason, the master MDT-object of the striped directory may lost its master LMV EA. If somebody created files under the master directly after the master LMV EA lost, then the LFSCK should NOT re-generate the master LMV EA, instead, it should change the broken striped directory as read-only to prevent further damage.
- test 31e: Re-generate the lost slave LMV EA for striped directory (1). For some reason, the first slave MDT-object of the striped directory (that resides on the same MDT as the master MDT-object) lost its slave LMV EA. The LFSCK should re-generate the slave LMV EA.
- test 31f: Re-generate the lost slave LMV EA for striped directory (2). For some reason, the non-first slave MDT-object of the striped directory (that resides on different MDT as the master MDT-object) lost its slave LMV EA. The LFSCK should re-generate the slave LMV EA.
- test 31g: Repair the corrupted slave LMV EA. For some reason, the stripe index in the slave LMV EA is corrupted. The LFSCK should repair the slave LMV EA.
- test 31h: Repair the corrupted shard's name entry. For some reason, the shard's name entry in the striped directory may be corrupted. The LFSCK should repair the bad shard's name entry.

Result

This test has been completed successfully and the results are recorded in the Implementation milestone:
http://wiki.opensfs.org/images/a/ad/LFSCK_MDT-MDTConsistency_Implementation.pdf

Single MDT Demonstration context

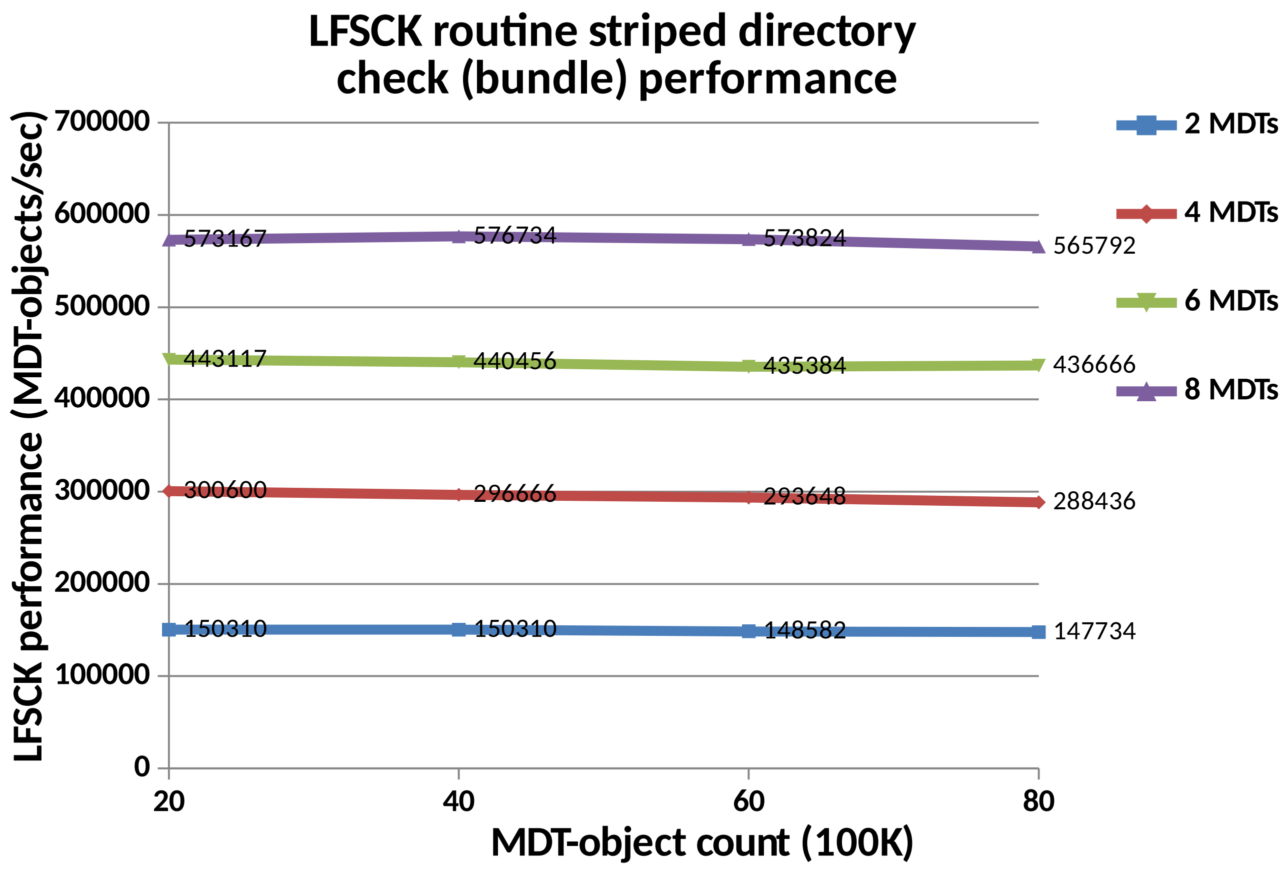
All tests require a populated directory on the file system. The directory will be created and populated with the following properties:

1. Create 'L' test root directories. 'L' is equal to the MDT count {2,4,6,8}. The directory in the root 'dir-X' is located MDT-X.
2. For each **MDT-X**, under its test root directory **dir-X**, create **M** sub-directories, where **M** = { 20, 40, 60, 80 }, for a maximum of 8 MDTs * 80 sub-directories per MDT = 640 directory trees.
3. Under each sub-directory, create 90,000 single-striped regular files, 12,500 local directories, 1000 2-linked files, 500 remote directories, and 500 full striped directories, for a total of 104,500 files and subdirectories per directory. This will create a maximum total of 640 * 90000 ~= 58M files and 640 * 13500 ~= 8.6M subdirectories.

Measure performance of LFSCK 3 against multiple, consistent MDTs.

This test provides a control benchmark for LFSCK 3 scanning. LFSCK 3 includes support for DNE striped and remote directories consistency checking (also known as MDT-MDT consistency checking). This test measures the scanning rate across multiple MDTs within striped directories is compares the result with expectations. The aggregate LFSCK scanning performance is expected to scale linearly as additional MDTs with objects are added to the filesystem.

Result



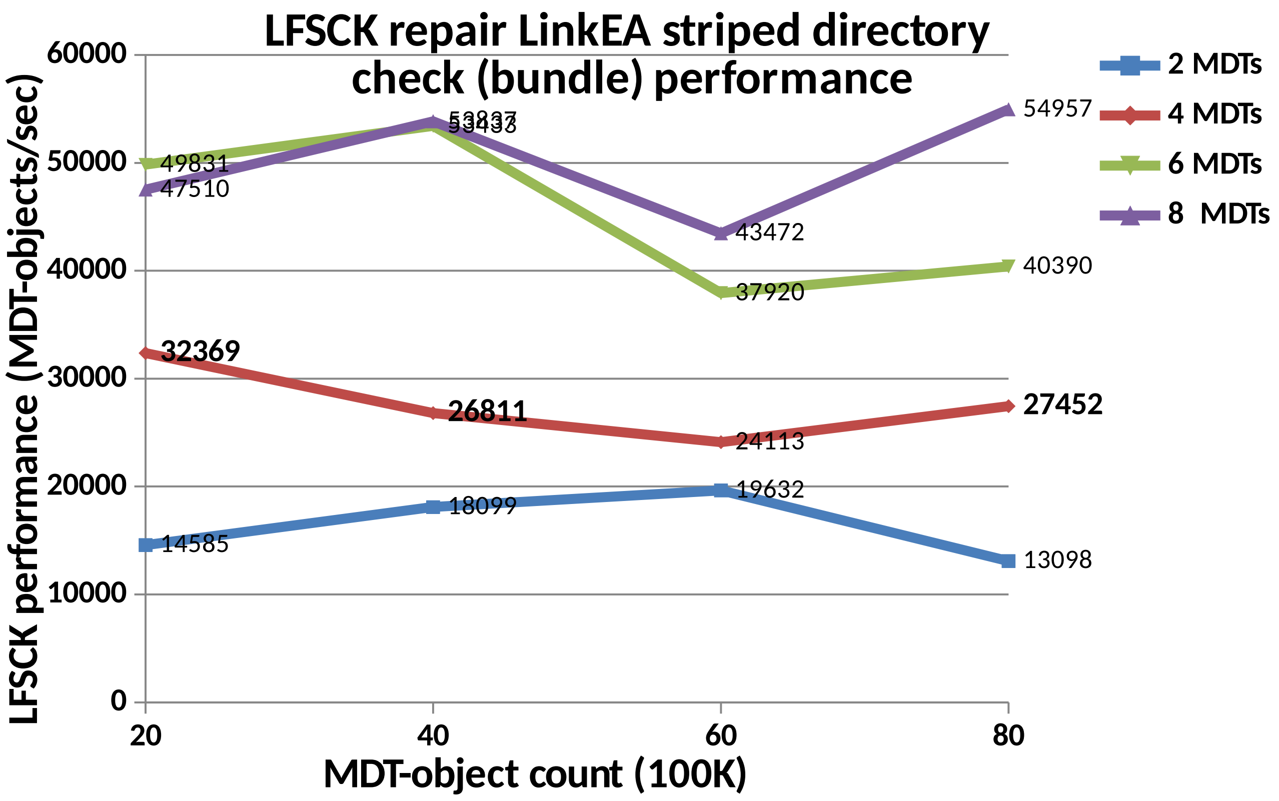
Checking a completely consistent file system performs favorably with previous results from the LFSCK project. For example, the Demonstration phase of LFSCK 2 measured a similar test on a single MDT at approximately at around 75,000 object a second. Here we see two MDTs at 150,000 object's a second, twice the number for a single MDT. As the MDT count is increased, performance apparently approaches linear scaling. This result is expected. As more MDTs are added, the count of high latency cross-MDT look-ups increases. These look-ups take longer than a local lookup result in a small trend away from linear scaling. A small reduction in performance is also apparently visible as MDT object count increases. The small decline in performance with additional MDT-objects is expected. This is because of experimental design: as the MDT object count increases the count of high latency cross-MDT look-up increases.

Measure performance of LFSCK 3 against a multiple MDTs with inconsistencies.

This test scans the full filesystem on all MDTs to look for inconsistencies in the filesystem namespace, including MDT-MDT inconsistencies. The intent of this test is to measure performance when the filesystem needs to repair a large number of inconsistencies.

In for this test the filesystem has been intentionally corrupted during the filesystem population. The specific corruption is a missing `link` xattr on each file in the filesystem. Inconsistencies are created using the `OBD_FAIL_LFSCK_NO_LINKEA` fault injection hook. The `link` xattr stores the backpointer from each inode to the directory name entry/entries for each link to the file. During scanning, the LFSCK traversal checks for each name entry in each directory whether a corresponding name entry exists in the `link` xattr. When LFSCK finds that no entry is present in the `link` xattr for each directory entry, the `link` xattr is updated with a new `{parent FID, filename}` entry for that directory entry. Files with multiple hard links will contain one entry in the `link` xattr for each link, subject to space availability in the `link` xattr.

Result

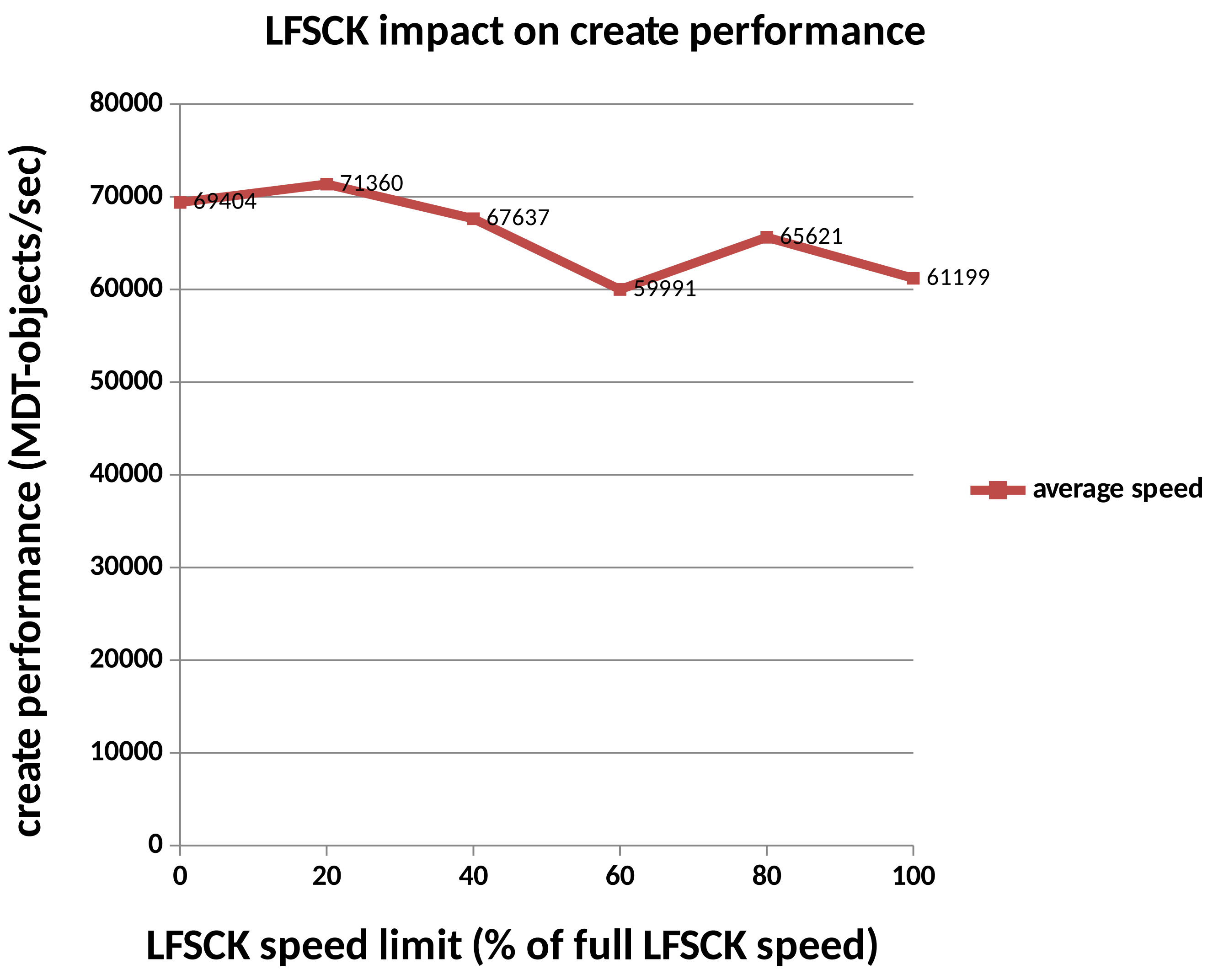


Checking an inconsistent file system performs favorably with results from the previous test of a consistent file system. Performance increases as MDTs are added appear to follow a linear trend. As larger file systems are tested, performance does not appear to slowdown. Measurements for this test apparently show increased variability across all observation points, compared to the previous test. This variation is expected as high-latency, typically random IO is needed to fix inconsistencies – as well as network lookups to resolve correct data values.

Impact of LFSCK on small file create performance on multiple MDT without inconsistencies

This test measures the additional load LFSCK 3 imposes on the MDS during a metadata-intensive application like workload. Online LFSCK includes a feature that allows the scanning rate to be limited. This feature is intended to enable an administrator to 'dial-back' the LFSCK scanning speed in a production environment to reduce or avoid impact on client metadata performance. This test provides a sweep of scanning rate measurements to give an administrator a feel for the performance change expected by choosing to reduce (or increase) the LFSCK scanning rate.

Result



Speed limit observations performs favorably with previous results from the LFSCK 2 project recorded in the Demonstration phase of LFSCK 2 milestone. Variation across the observations is higher than expected and apparently greater than observed during a similar test for LFSCK 2. Multiple observations of each data point were made for this experiment. A high level of variation for between consecutive observations was observed. It is thought that this variation could be the result of client-side FID allocation.

Conclusion

LFSCK 3: MDT-MDT consistency has successfully completed both functional Acceptance and Performance tests. The performance results recorded herein illustrate performance expectations are met or exceeded during online operation and under load. In addition, LFSCK 3 has been shown to meet or exceed expectations running in a multiple MDT environment.

Appendix A: OpenSFS Functional Test Cluster specification

client

- (2) Intel E5620 2.4GHz Westmere (Total 8 Cores)
- (1) 64GB DDRIII 1333MHz ECC/REG - (8x8GB Modules Installed) * (1) On Board Dual 10/100/1000T Ports
- (8) Hot Swap Drive Bays for SATA/SAS
- (6) PCi-e Slots 8X
- (3) QDR 40GB QSFP to QSFP iB Cables
- (3) Mellanox QDR 40GB QSFP Single Port

OSS server

- (1) Intel E5620 2.4GHz Westmere (Total 8 Cores)
- (1) 32GB DDRIII 1333MHz ECC/REG - (8x8GB Modules Installed) * (1) On Board Dual 10/100/1000T Ports
- (1) On Board VGA
- (1) On Board IPMI 2.0 Via 3rd. Lan
- (1) 500GB SATA Enterprises 24x7
- (1) 40GB SSD OCZ SATA
- (8) Hot Swap Drive Bays for SATA/SAS
- (6) PCi-e Slots 8X
- (3) QDR 40GB QSFP to QSFP iB Cables
- (3) Mellanox QDR 40GB QSFP Single Port

MDS server

- (1) Intel E5620 2.4GHz Westmere (Total 8 Cores)
- (1) 32GB DDRIII 1333MHz ECC/REG - (8x8GB Modules Installed) * (1) On Board Dual 10/100/1000T Ports
- (1) On Board VGA
- (1) On Board IPMI 2.0 Via 3rd. Lan
- (1) 500GB SATA Enterprises 24x7
- (1) 40GB SSD OCZ SATA
- (8) Hot Swap Drive Bays for SATA/SAS
- (6) PCi-e Slots 8X
- (3) QDR 40GB QSFP to QSFP iB Cables
- (3) Mellanox QDR 40GB QSFP Single Port