ORNL/TM-2015/608

Lustre Distributed Name Space (DNE) Evaluation at the Oak Ridge Leadership Computing Facility (OLCF)

James S. Simmons
Dustin Leverman
Jesse Hanley
Sarp Oral

August 22, 16

OAK RIDGE NATIONAL LABORATORY

ORNL/TM-2015/608

Center for Computational Sciences

Lustre Distributed Name Space (DNE) Evaluation at the Oak Ridge Leadership Computing Facility (OLCF)

James S. Simmons
Dustin Leverman
Jesse Hanley
Sarp Oral

Date Published: August 22, 16

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-BATTELLE, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

ABSTRACT

This document describes the Lustre Distributed Name Space (DNE) evaluation carried at the Oak Ridge Leadership Computing Facility (OLCF) between 2014 and 2015. DNE is a development project funded by the OpenSFS, to improve Lustre metadata performance and scalability. The development effort has been split into two parts, the first part (DNE P1) providing support for remote directories over remote Lustre Metadata Server (MDS) nodes and Metadata Target (MDT) devices, while the second phase (DNE P2) addressed split directories over multiple remote MDS nodes and MDT devices. The OLCF have been actively evaluating the performance, reliability, and the functionality of both DNE phases. For these tests, internal OLCF testbed were used. Results are promising and OLCF is planning on a full DNE deployment by mid-2016 timeframe on production systems.

1. INTRODUCTION

The Oak Ridge National Laboratory Leadership Computing Facility (OLCF) at the Oak Ridge National Laboratory (ORNL) is currently the home of the world's second-fastest supercomputer named Titan. Any large-scale computational resource needs to have a storage system that is designed to ingest data at rates fast enough as to not waste excessive computational cycles. There are many aspects to being able to inject data at high speeds to a storage system and it is mainly dependent on what type of I/O the computational codes running on the supercomputer are doing.

In the case of Titan, at any given time there could be multiple jobs running, and that is only one of the platforms (e.g. computational, visualization, post-processing, or data transfer) operating in the OLCF. Our approach to high-speed data storage in the OLCF was to provide a large Lustre-based general-purpose storage system that is globally mount-able by all systems that need access. This file system is named Atlas, and is broken into two distinct namespaces: Atlas1 and Atlas2. In aggregate, these two storage systems have a capacity of 30 PB and a performance throughput of over 1 TB/sec.

Both Atlas namespaces are built using the Lustre technology. Lustre is an open-source parallel file system technology that is currently used over 70% of the Top500 systems around the world. Lustre is a high-performance file system scalable to tens of thousands of clients. Although Lustre can achieve very high file I/O performance numbers (e.g. 3 TB/s @ Rikken, Japan, +1 TB/s @ Titan, OLCF and BlueWaters, NCSA), it supported only one metadata server per file system namespace until recently. This was one of the biggest limitations Lustre had. To alleviate this problem, OpenSFS contracted Intel to develop a solution (http://wiki.opensfs.org/Contract_SFS-DEV-001).

The Distributed Namespace (DNE) development effort started in 2011 and targeted improving the metadata server scalability. The DNE effort was split up into two development phases. Phase 1 (DNE P1) provided metadata support for directories distributed over remote metadata server (MDS) nodes and metadata target (MDT) devices. This feature was released to public in Lustre version 2.4 in 2012. In Phase 2 (DNE P2), the capability for splitting up a single directory over remote MDS nodes and MDT devices was added. This will be released to public in Lustre version 2.8.

This document summarizes OLCF's efforts in evaluating the Lustre DNE feature. The OLCF's plan is deploy DNE P2 into production by the second half of 2016, however, OLCF tests with DNE P2 is not yet complete and this document will updated with new results in the near future.

2. METADATA CONTENTION

Lustre file system can scale up to tens of thousands clients. However, Lustre, as of version 2.4, supported only a single metadata server (MDS) per file system name space. As client numbers continue to increase, a single MDS quickly created a performance and scalability bottleneck. DNE is a Lustre development project, funded by OpenSFS, spreading the Lustre file system metadata namespace horizontally and vertically over multiple MDS nodes and metadata targets (MDTs). Here, horizontally and vertically spreading means distributing the metadata workload over multiple MDTs, over directories (horizontally) and per directory (vertically). This is akin to spreading the Lustre file system I/O over multiple object storage server (OSS) nodes and object Storage Targets (OSTs). This enables namespace size and metadata throughput to be increased by the addition of MDTs. In addition, DNE enables an administrator to allocate metadata resources to specific directories within the file system.

3. DISTRIBUTED NAMESPACES (DNE) ON LUSTRE

The DNE contract (http://wiki.opensfs.org/Contract_SFS-DEV-001) was awarded to Whamcloud (now Intel HPDD) in 2011, and defined the Lustre development for three projects and eight subprojects. These three projects were issued as a single contract since they were complementing each other. Overall, these three development efforts effectively remove the metadata contention problem on Lustre. Of these three projects, Project 1 focused on improving the single server metadata performance and provided SMP node affinity and parallel directory support. The second project (i.e. DNE) targeted distributing the metadata load on remote and striped directories. The third project was architected to improve the Lustre file system checker to provide support in light of the first projects.

DNE P1 enabled deployment of multiple MDTs on multiple MDS nodes and Lustre sub-directories were now distributed over multiple metadata targets (MDTs). Sub-directory distribution was defined as an administrative operation and was executed using the Lustre specific *lfs mkdir* command. Figure 1 illustrates the Lustre remote directory and sub-directory concepts. By choice, it was functionality restricted to only specially created directories to remote MDTs (remote directories with their parent directory on MDT0; remote directories with parent directories on other MDTs was made possible with administrator override) and it included capabilities such as user tools to create directories on a specific remote MDT and to display on which MDT a directory or file is located.

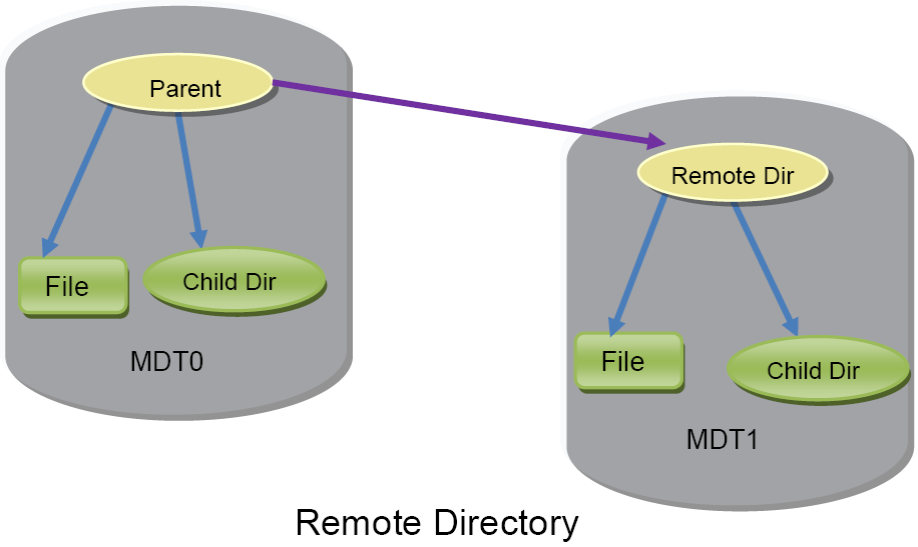


Figure 1: DNE P1 Remote Directories (Courtesy of OpenSFS)

DNE P2 will provide Lustre the capability to create striped directory in a similar fashion as bulk I/O is distributed across OSTs. The OSS bulk I/O packets are distributed in a round robin fashion with applied weights to help keep OST usage nearly the same across the namespace. In the case of meta data the distribution is based on a hash calculated for the file in interest. You can control the hashing done for a striped directory with *lfs setdirstripe* using the *–t or --hash-type option*. (Currently only two options are available, *all_char* which is sum of ASCII characters modulo number of MDTs, or *fnv_1a_64* that is the Fowler-Noll-Vo (FNV-1a) algorithm. From the previous example you can see the user interface has moved from DNE P1 *lfs mkdir [option]* to using *lfs setdirstripe [option]* for DNE P2. Much like DNE P1 directories creation can be limited to personnel with root privileges. By default, only directories on the very first MDT can contain directories that are not on the same MDT. Figure 2 illustrates this Lustre striped directory concept, where a striped directory is defined as the contents of a given directory are

distributed over multiple MDTs. Looking back to Figure 1 you will see a similar limitation with DNE P1 in which data from the root remote directory is cached on the first MDT.

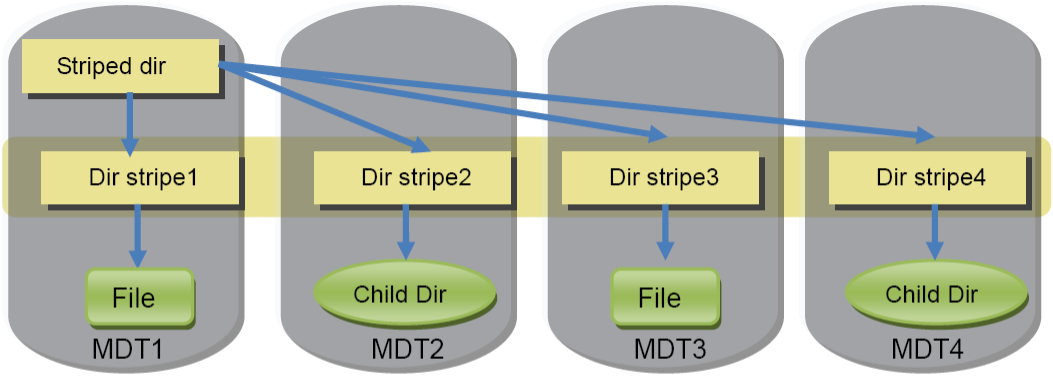


Figure 2: DNE P2 Stripe Directories (Courtesy of OpenSFS)

In either case we can see the limitation is that MDT1 contains a disproportionate amount of data. This is due to fact that, the root is MDT1 and it is not random. In testing, when the file and directory count reached high enough the first MDT filled rapidly. This was exacerbated since the first MDT was the smallest in the test cluster. For DNE P2 it is possible to create remote directories with its parent directory located on any MDT by enabling the remote_dir feature. This can be done by issuing the follow command:

lctl set_param mdt..enable_remote_dir=1*

to all available MDS servers. Another feature available in DNE P2 is the ability to allow non root users to create remote directories. Currently you can use the default setup up of only allowing root or you can allow a single specific group to create a remote directory which handles the case of specific non root administrative accounts. Lastly you can allow all groups. The way to manage this feature is by issuing the following command to all MDS servers:

lct set_param mdt..enable_remote_dir_gid="group_id"*

To allow any user to create and remove DNE2 directories set "group_id" to -1. So far all the functionality discussed has pertained specifically to DNE2. More functionality exists for the *lfs setdirstripe* command, which was modeled after the same functionality seen with *lfs setstripe* (used to manage I/O bulk data placement on the OSS servers). This was to done to make it easy for new users to learn to use this tool quickly since they are already familiar with *lfs getstripe*. The *lfs setdirstripe,* like its counterpart, allows setting the striping count and striping index. The striping index can be used to control which MDT is to be the first to store meta data information. This is helpful in avoiding MDTs that lack enough disk space. The stripe count determines how many MDTs will be used to store the file's meta data for a remote directory.

One last feature added to DNE2 is the inheritance of the directory striping to subdirectories. To enable this feature one uses the -D option available to *lfs setdirstripe.* This changes the behavior of how DNE2 handles data contained in the subdirectory tree. With the inheritance options any subdirectory

created inside a DNE2 striped directory will have the same striping pattern as its parent. This means all the files created in that new subdirectory will be hashed across all the MDT mapped to the parent DNE2 directory. Each new subdirectory created will have it meta data also hash across all the MDTs which will have a negative impact on directory operation performance. This is demonstrated in the data presented in this documentation. This is not the default behavior for this reason. Looking at figure 2 carefully you can see the default behavior is that newly created subdirectories of a DNE2 stripe directory will be mapped to one specific MDT based on the hashing algorithm just as the files are done. For example in figure 2 the two created subdirectories are located on MDT2 and MDT4. This implies that all files created in those two subdirectories will be located only on MDT2 or on MDT4 respectively. You can think of these subdirectories as DNE1 class directories. The difference is that unlike DNE1 these new subdirectories will not have a duplicate inode on MDT1. Also unlike DNE1 it is possible for a normal user to create a new subdirectory if you enable the remote_dir_gid parameter is set to -1.

4. DNE PHASE 1 EVALUATION

Our test environment consisted Arthur, which is a single Cray XK7 cabinet with a total of 35 nodes of which 20 nodes were of the compute class. Arthur at the time of DNE P1 tests were running Cray CLE 5.2.UP02 operating system. The Lustre version on clients was 2.5.3. 16 separate x86_64 systems were configured as Lustre MDS nodes. Servers were running Lustre version 2.5.3. Each node had a single Toshiba MBF2600RC Serial Attached SCSI SSD device. These were configured as MDTs, one per MDS.

We used an in-house modified version of the mdtest benchmark version 1.8.3. Our modification allowed mdtest to use precreated directories, which was essential in testing the DNE P1. In the standard mdtest each iteration of test creates a new directory #test-dir.'iteration' which will always be the non-DNE case. To work around this case mdtest was modified to use detect the already existing test directories. The second change is mdtest runs it test against each test directory one at a time but since each directory is located on a single MDT with each being on a different server we can not evaluate the performance gain in that setup. So the second modification to mdtest was to allow running all test directory iterations at the same time in parallel. The mdtest benchmark execution script can be found in Appendix A. The modified mdtest source code changes will be merged back into the publicly available mdtest soon.

For DNE P1 testing, we used 1,000 files per directory as a constant MDS directory load. Tests were carried out on October 2014. We used 19 client nodes on Arthur with 32 tasks per each, 608 tasks total for the tests. Each test were repeated for 10 iterations. A test sample output is given below:

```
-- started at 10/21/2014 14:22:23 --
mdtest-1.8.3 was launched with 608 total task(s) on 19 nodes
Command line used: /lustre/sultan/stf008/scratch/jsimmons/mdtest -i 10 -I 1000 -v -u -d
/lustre/sultan/stf008/scratch/jsimmons/test_mdt0@/lustre/sultan/stf008/scratch/jsimmons/test_mdt1
Path: /lustre/sultan/stf008/scratch/jsimmons
FS: 797.1 TiB   Used FS: 0.0%   Inodes: 398.7 Mi   Used Inodes: 0.0%
608 tasks, 608000 files/directories
Operation               Duration              Rate
---------               --------              ----
* iteration 1 *
   Tree creation     :      0.021 sec,     46.915 ops/sec
   Directory creation:     50.762 sec,  11977.446 ops/sec
   Directory stat    :      5.075 sec, 119803.236 ops/sec
   Directory removal :     17.270 sec,  35206.154 ops/sec
   File creation     :     18.693 sec,  32525.254 ops/sec
   File stat         :     23.776 sec,  25571.721 ops/sec
   File removal      :     38.891 sec,  15633.324 ops/sec
   Tree removal      :      0.053 sec,     18.905 ops/sec
* iteration 2 *
   Tree creation     :      0.016 sec,     63.646 ops/sec
```

```
   Directory creation:     53.253 sec,  11417.247 ops/sec
   Directory stat    :      4.995 sec, 121719.575 ops/sec
   Directory removal :     18.486 sec,  32890.048 ops/sec
   File creation     :     16.301 sec,  37297.574 ops/sec
   File stat         :     23.698 sec,  25656.142 ops/sec
   File removal      :     37.839 sec,  16067.906 ops/sec
   Tree removal      :      0.103 sec,      9.747 ops/sec
 * iteration 3 *
   Tree creation     :      0.048 sec,     20.784 ops/sec
   Directory creation:     56.738 sec,  10715.979 ops/sec
   Directory stat    :      5.006 sec, 121445.960 ops/sec
   Directory removal :     16.371 sec,  37137.929 ops/sec
   File creation     :     22.764 sec,  26708.523 ops/sec
   File stat         :     23.661 sec,  25696.630 ops/sec
   File removal      :     36.232 sec,  16780.610 ops/sec
   Tree removal      :      0.086 sec,     11.603 ops/sec
 * iteration 4 *
   Tree creation     :      0.030 sec,     33.109 ops/sec
   Directory creation:     65.516 sec,   9280.225 ops/sec
   Directory stat    :      5.034 sec, 120789.771 ops/sec
   Directory removal :     10.381 sec,  58570.283 ops/sec
   File creation     :     31.259 sec,  19450.693 ops/sec
   File stat         :     23.761 sec,  25587.876 ops/sec
   File removal      :     37.799 sec,  16085.099 ops/sec
   Tree removal      :      0.092 sec,     10.826 ops/sec
 * iteration 5 *
   Tree creation     :      0.126 sec,      7.917 ops/sec
   Directory creation:     56.259 sec,  10807.185 ops/sec
   Directory stat    :      4.988 sec, 121891.029 ops/sec
   Directory removal :     14.342 sec,  42393.831 ops/sec
   File creation     :     19.169 sec,  31717.449 ops/sec
   File stat         :     23.509 sec,  25862.805 ops/sec
   File removal      :     38.008 sec,  15996.705 ops/sec
   Tree removal      :      0.100 sec,      9.985 ops/sec
 * iteration 6 *
   Tree creation     :      0.024 sec,     41.264 ops/sec
   Directory creation:     50.711 sec,  11989.447 ops/sec
   Directory stat    :      5.050 sec, 120398.587 ops/sec
   Directory removal :     13.716 sec,  44327.294 ops/sec
   File creation     :     23.142 sec,  26272.491 ops/sec
   File stat         :     22.998 sec,  26437.275 ops/sec
   File removal      :     31.862 sec,  19082.082 ops/sec
   Tree removal      :      8.808 sec,      0.114 ops/sec
 * iteration 7 *
   Tree creation     :      0.023 sec,     44.131 ops/sec
   Directory creation:     49.275 sec,  12338.823 ops/sec
   Directory stat    :      4.956 sec, 122684.801 ops/sec
   Directory removal :     17.558 sec,  34627.587 ops/sec
   File creation     :     17.538 sec,  34667.561 ops/sec
   File stat         :     23.453 sec,  25924.256 ops/sec
   File removal      :     38.894 sec,  15632.386 ops/sec
   Tree removal      :      0.071 sec,     14.008 ops/sec
 * iteration 8 *
   Tree creation     :      0.027 sec,     37.594 ops/sec
   Directory creation:     54.556 sec,  11144.496 ops/sec
   Directory stat    :      5.004 sec, 121512.774 ops/sec
   Directory removal :     11.720 sec,  51878.271 ops/sec
   File creation     :     18.477 sec,  32905.459 ops/sec
   File stat         :     24.315 sec,  25005.094 ops/sec
   File removal      :     44.397 sec,  13694.737 ops/sec
   Tree removal      :      0.134 sec,      7.488 ops/sec
 * iteration 9 *
   Tree creation     :      0.027 sec,     37.394 ops/sec
   Directory creation:     53.182 sec,  11432.347 ops/sec
   Directory stat    :      5.063 sec, 120084.463 ops/sec
   Directory removal :     17.930 sec,  33910.568 ops/sec
   File creation     :     24.783 sec,  24532.989 ops/sec
   File stat         :     23.972 sec,  25363.104 ops/sec
   File removal      :     36.834 sec,  16506.375 ops/sec
   Tree removal      :      0.063 sec,     15.902 ops/sec
```

```
* iteration 10 *
   Tree creation     :      0.018 sec,     57.077 ops/sec
   Directory creation:     59.193 sec,  10271.559 ops/sec
   Directory stat    :      5.028 sec, 120923.844 ops/sec
   Directory removal :     16.536 sec,  36768.123 ops/sec
   File creation     :     24.106 sec,  25222.394 ops/sec
   File stat         :     23.863 sec,  25478.979 ops/sec
   File removal      :     39.525 sec,  15382.738 ops/sec
   Tree removal      :      0.049 sec,     20.433 ops/sec

SUMMARY: (of 10 iterations)
   Operation               Max        Min        Mean     Std Dev
   ---------               ---        ---        ----     -------
   Directory creation:  12349.796   9283.845  11145.773    869.879
   Directory stat    : 123904.353 120985.660 122593.489    887.006
   Directory removal :  58571.575  32890.567  40771.867   8140.125
   File creation     :  37338.848  19460.639  29148.300   5243.593
   File stat         :  26506.056  25039.761  25720.620    360.899
   File removal      :  19082.413  13694.820  16086.342   1275.494
   Tree creation     :     63.646      7.917     38.983     15.380
   Tree removal      :     20.433      0.114     11.901      5.570

-- finished at 10/21/2014 14:49:01 --
Application 183 resources: utime ~174191s, stime ~9984s, Rss ~8752, inblocks ~123436, outblocks ~259931
```

Figure 3 shows DNE P1 testing for directory IOPs rates with respect to increasing MDS nodes. As can be seen, directory deletions quadruple the performance from 1 to 5 MDS nodes, then the performance drops around 25% and stays flat. For directory stat operations, the performance gain quickly diminishes after 3 MDS nodes. For directory create operations, the ramp up is quite slow compared to deletions and stats, and best performance is obtained at 13 MDS nodes. From these results, OLCF determined that the best deployment options were to limit the number of MDS nodes to four, if directory operation rates were the primary performance concern.

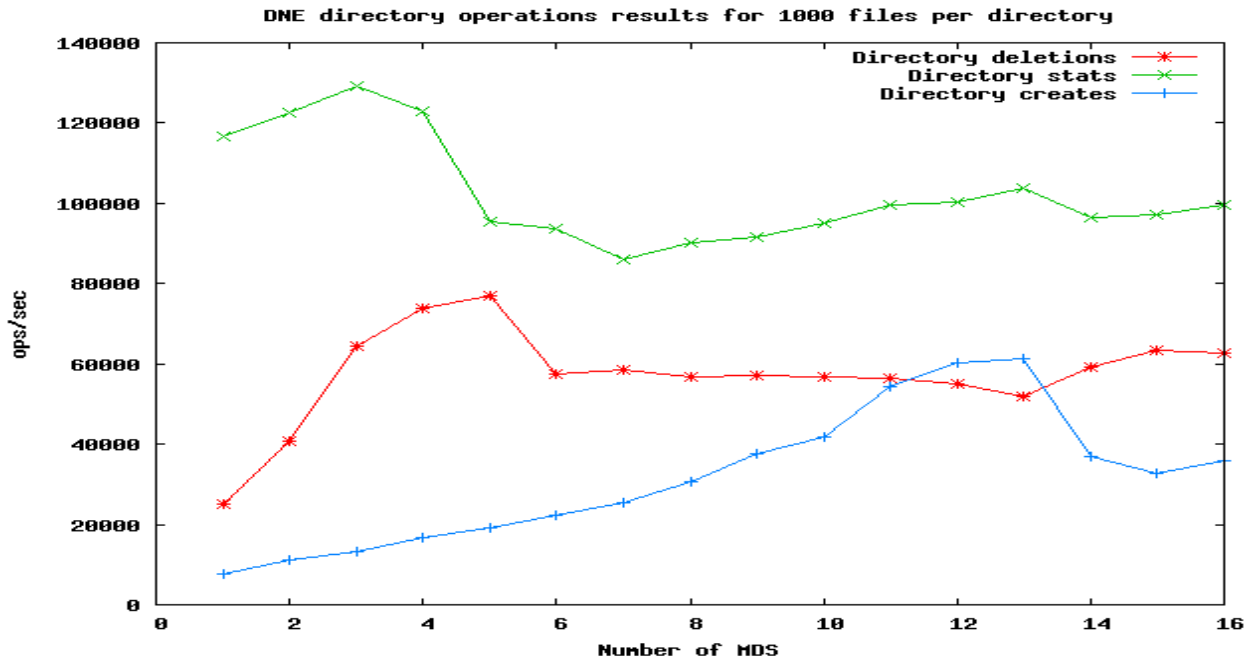


Figure 3: DNE P1 directory test results

Figure 4 shows our DNE P1 test results with file operations. As can be seen, file deletion operations scale up nicely up to 15 MDS nodes, where increasing number of MDS nodes hurts file stat operation rates. File creation performance doubles up until five MDS nodes only.

Combining DNE P1 test results for directory and file operations, best rule of thumb was determined as 4 MDS nodes per namespace for production deployments.

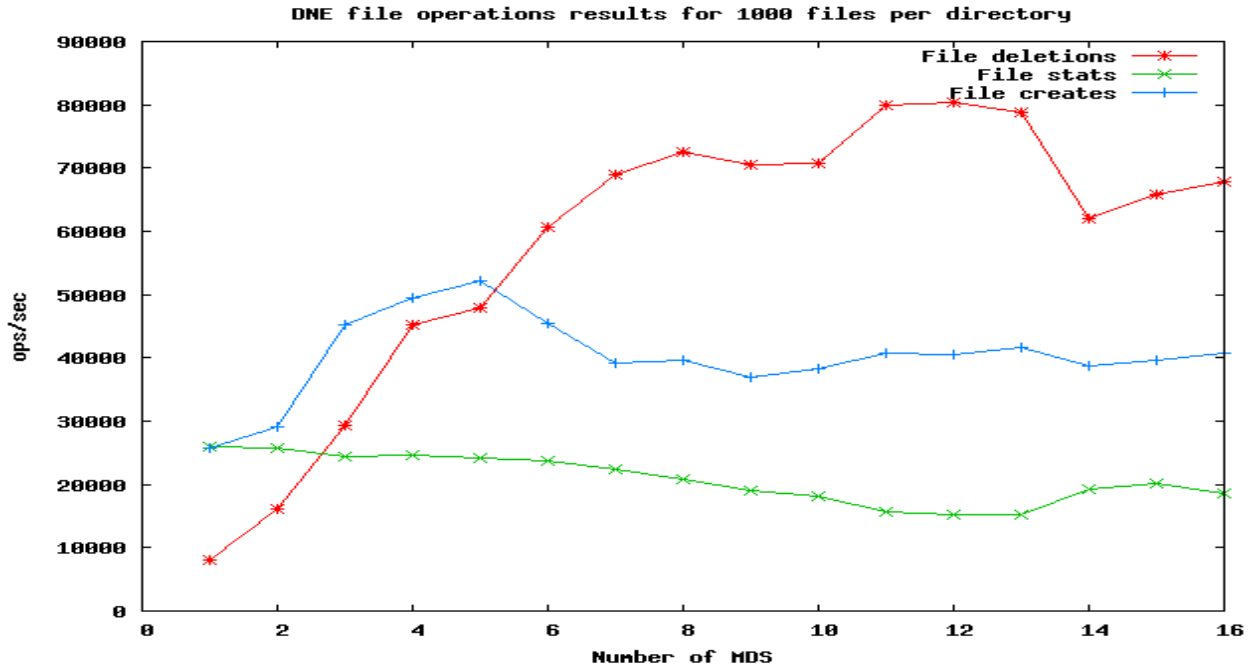


Figure 4: DNE P1 file test results

5. DNE PHASE 2 EVALUATION

Our test environment consisted of Arthur, which is a single Cray XK7 cabinet with a total of 35 nodes of which 20 nodes were of the compute class. Arthur at the time of DNE P2 tests was running a Cray CLE 5.2.UP04 operating system. The Lustre version on clients was a specially built 2.8.0 client. Testing was performed at various stages of development of the DNE2 work to ensure proper behavior as well improving performance. The results presented here represents the performance behavior that will be encountered in our production system once upgraded to the official Lustre 2.8.0 release. On the server side, sixteen separate x86_64 systems were configured as Lustre MDS nodes. Servers were running the same Lustre 2.8.0 version as the clients. Each node has a single Toshiba MBF2600RC Serial Attached SCSI SSD device. These were configured as MDTs, one per MDS.

For the DNE P2 testing, we chose ten thousand and one hundred thousand files per directory to create a constant MDS directory load. Additionally, we ran tests using one million files per directory when time permitted. The focus of the testing was to determine a client scaling profile for the case of one, two, four, eight and sixteen MDT/MDS nodes. To duplicate the default setup in our production system, all directories were striped across four OSTs with each OST located on a different OSS server. As we scaled the testing the client threads were distributed evenly across all twenty compute nodes on Arthur. At every new job launch of mdtest four additional threads spawned on four compute nodes. The test concluded when four hundred threads, 20 threads on 20 clients, were completed. Four threads made an optimal

mapping for our MDS striping cases of 1,2, 4, 8 and possibly 16. Five iterations were done for each individual test. The number of files per directory remained the same independent of the number of client threads. With increasing thread count this translates into less file operations per thread.

The file system in all cases was configured to allow any user to create a directory with the desired MDS striping pattern. This gave us the flexibility to execute mdtest as a non-root user. The first set of tests for our evaluation was done for the DNE2 inheritance case. To force inheritance of the MDS striping to newly created subdirectories the –D option was run on the top most level of the testing directory before mdtest was executed. With this setup any subdirectories created by mdtest automatically have the same MDS striping pattern as the top level directory created for testing. Since all new subdirectories would be created with the exact same MDS striping pattern, a modified mdtest version was not need. Version 1.9.3 of mdtest was used for our non-inheritance testing. In the case of non-inheritance, all files created in any subdirectory would be hashed across the selected number of metadata servers. This allows a single shared directory for file operation performance evaluation.

In the next round of testing the top test directory's striping properties were not inherited to any subdirectories created. As before, non-root users had the ability to create directories of various MDS stripe count. For the case of DNE2 striped directories any files created still will be hashed across the MDS nodes. This will also be true for any created subdirectories. For example, if directory A is striped across four metadata servers then four created subdirectories would each be mapped to a different metadata server. Therefore, directory B would be mapped to MDS 1, directory C mapped to MDS 2 and so on. Any files created in one of those subdirectories will be mapped to one specific MDS. In the case of our example all files created in subdirectory C will be present only on the second MDS. Running mdtest with the unique directory option allows file operations to occur in parallel on a single node, maximizing performance. With this option each MPI process will be mapped to a unique subdirectory that in turn map to one specific metadata server. Using an example of four MPI processes on one node for the four MDS case shows that each unique directory created by each MPI process will map to a distinct MDS node.

Unlike the earlier inheritance testing, mdtest had to be modified to work with our requirements for this next set of test cases. When mdtest launches, it creates a new subdirectory for each iteration executed in the requested parent directory. In each subdirectory created per iteration, batches of new subdirectories are created for the directory operations testing as well for file operations testing since the unique option was requested. Based on the DNE2 top level directory stripe settings, each iteration's subdirectory will be mapped to some unique MDS node. All subdirectories of the iteration's directory created will be mapped to the exact same MDS as the iteration's subdirectory. Since all iterations run sequentially it will behave as if only one metadata server is being used at time. This will cancel out any performance benefit from DNE2 striping.

The way to work around this limitation was to add the ability to set the DNE2 striping on the iteration subdirectory. The code modifications were done against the latest mdtest sources from the git repository hosted at LLNL. These additional changes to mdtest will be pushed to the LLNL git repository in the near future. The ability to run this class of testing can then be available with the next release of mdtest. With the iteration directory striped across multiple metadata servers, any subdirectories created inside the iteration directories will be mapped to one unique metadata server. Any files created in one of the subdirectory will automatically map to one MDS as well. The unique option for testing file operations causes each MPI process on a node to create a new subdirectory mapped to a different MDS server. This will in turn maximize the performance potential of DNE2 striping allowing multiple parallel file operations on a single node. For our directory operation testing, having DNE2 striping on the iteration subdirectories all the subdirectories created will much like the file creation case be mapped to different unique MDS. This allows parallel directory creation, deletion, and stats collection from a single node.

5.1 FILE PERFORMANCE PROFILE FOR INHERTIED CASE

First, we examine the ten thousand files per directory case. From that data one can see excessive noise due to the small file sample size in each directory. With the latter results of larger file counts a clearer picture will merge for the results. For this setup mdtest was using a single shared directory. All the files created were hashed across some number of MDTs depending on which test was executed.

5.1.1 Ten thousand files per single shared directory

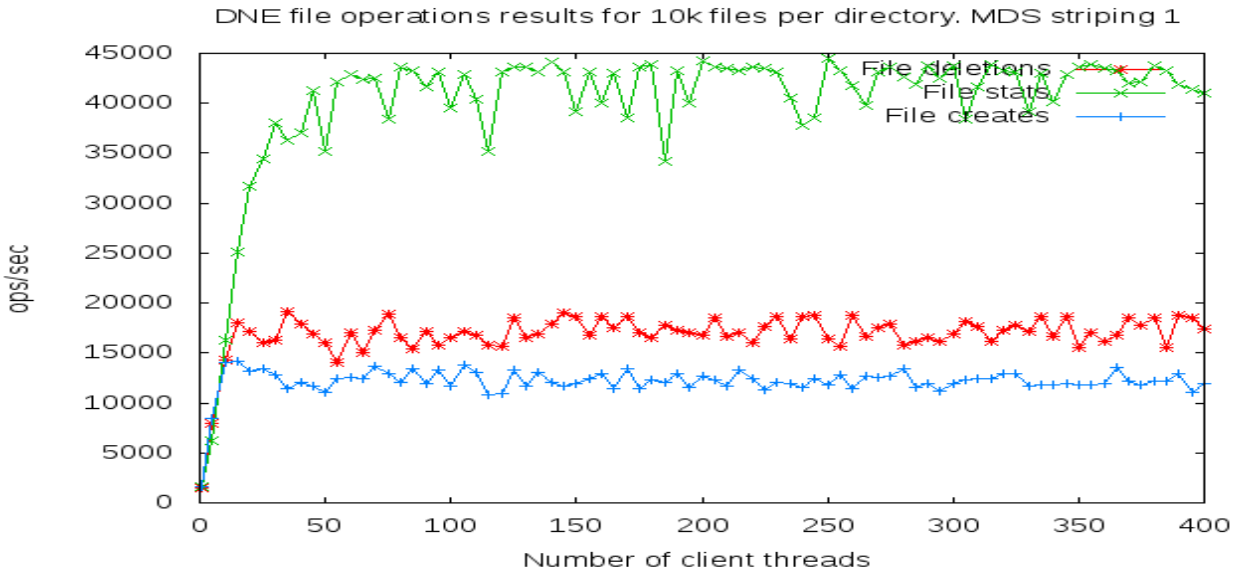


Figure 5: DNE P2 inheritance 10k file test results for single MDS

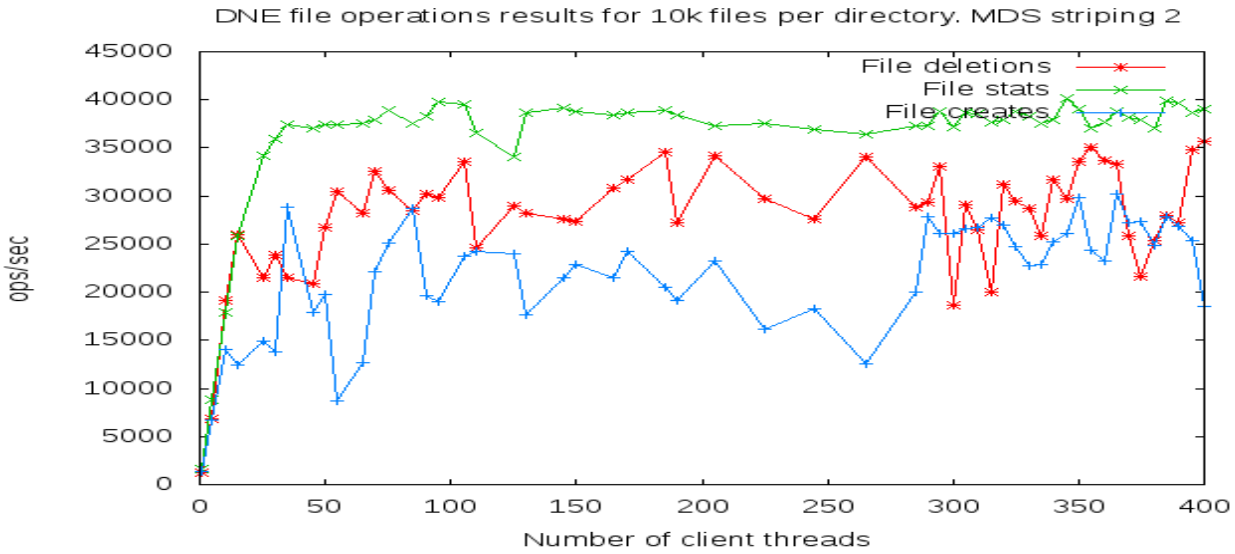


Figure 6: DNE P2 inheritance 10k file tests results using 2 MDTs

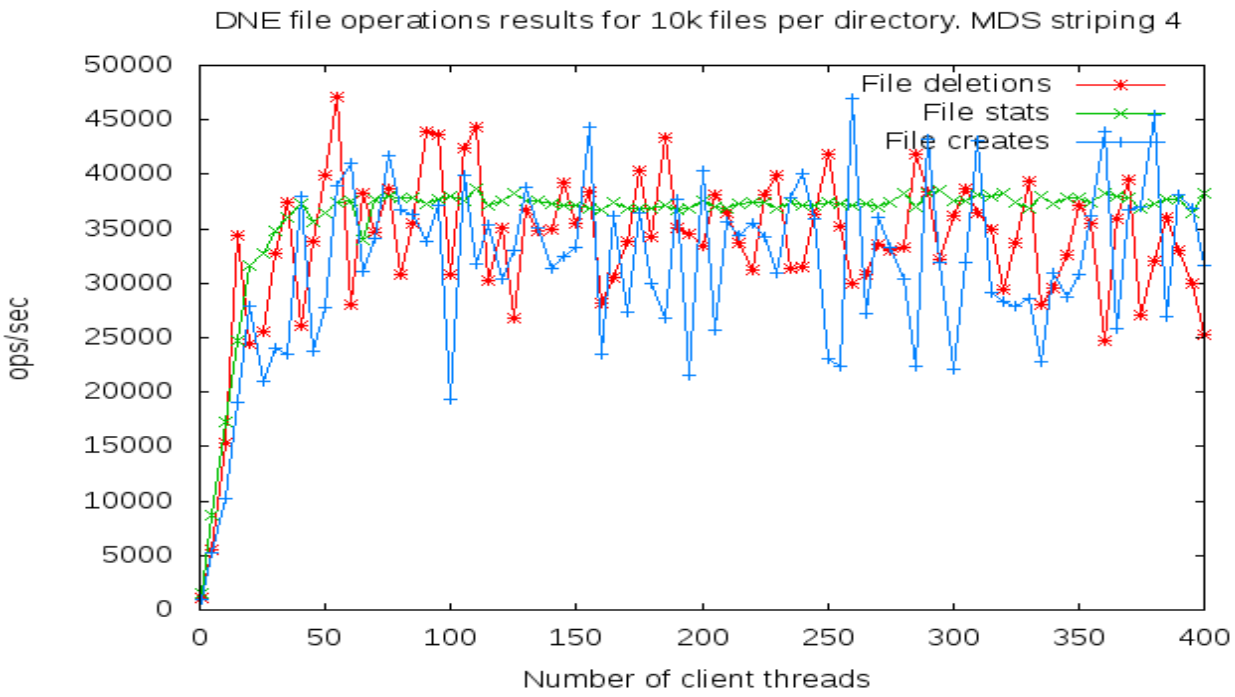


Figure 7: DNE P2 10k file test results using 4 MDTs

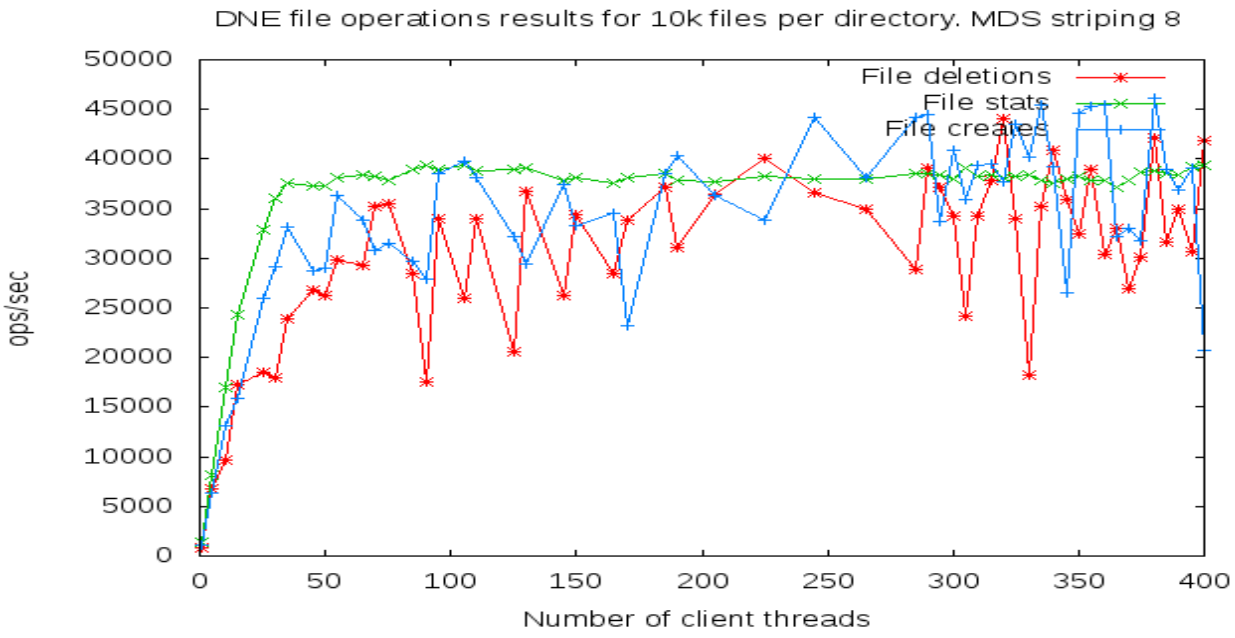


Figure 8: DNE P2 inheritance 10k file test results using 8 MDTs

From the ten thousand files per directory case we approached saturation below fifty nodes. Examining the raw data at around 35 nodes the file operations have reached their upper boundary. Two threads per node run across the twenty computes nodes will reach the file operations maximum threshold of our current testing hardware setup. Distributing files across all MDTs does nothing to improve file stats in a single shared directory. As for file creation and removal, improvements are observed. We see as the number of

MDTs increase, the behavior of file creation and deletion approach the behavior of the stating of the files. Little benefit is observed with greater than four MDTs.

5.1.2 One hundred thousand files per single shared directory

The next logical step is to increase the number of files per single shared directory to ensure that the file operations behavior scales in an appropriate way. Examination of the following graphs demonstrates that like the ten thousand file case that DNE P2 distributed files in a single shared directory brings little benefits to stating files or file removal. In fact, using more than one MDT has a small performance penalty for stating files. Using more than 4 MDTs gives little advantage and in the case of file deletion the 8 MDT case loses some of the performance gain of the 4 MDT case.

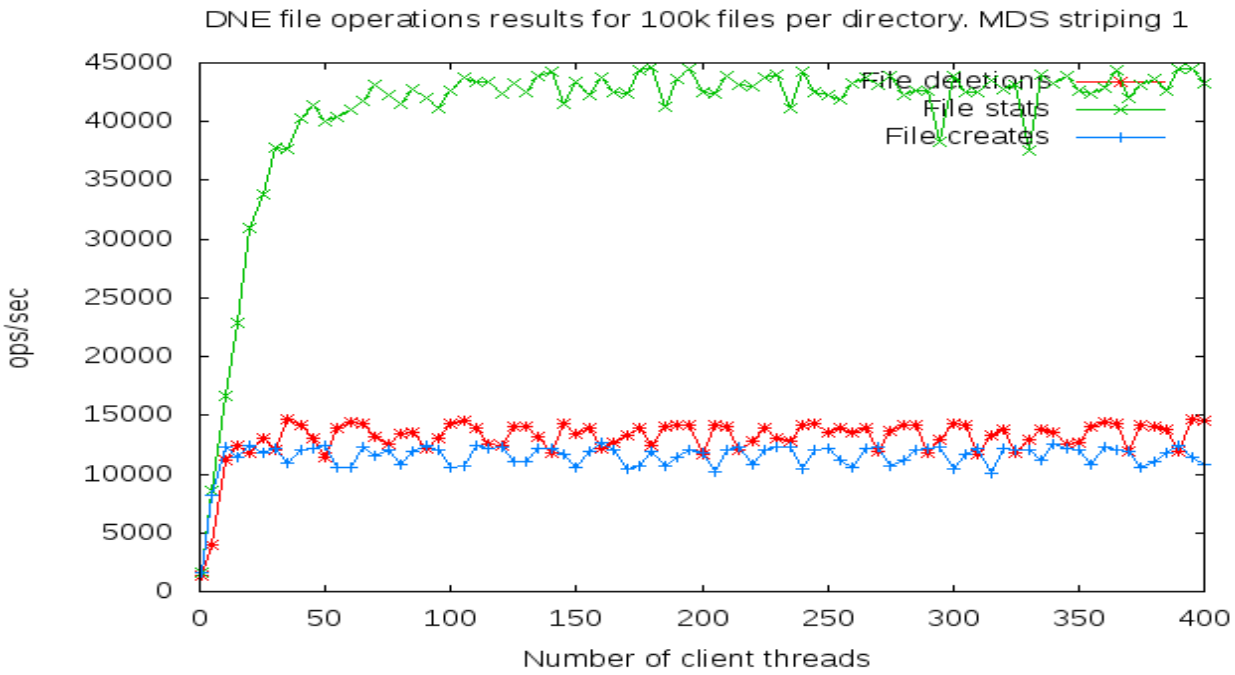


Figure 9: DNE P2 inheritance 100k files using a single MDT

Figure 10: DNE P2 inheritance 100k files using two MDTs

Figure 11: DNE P2 inheritance 100k files using 4 MDTs

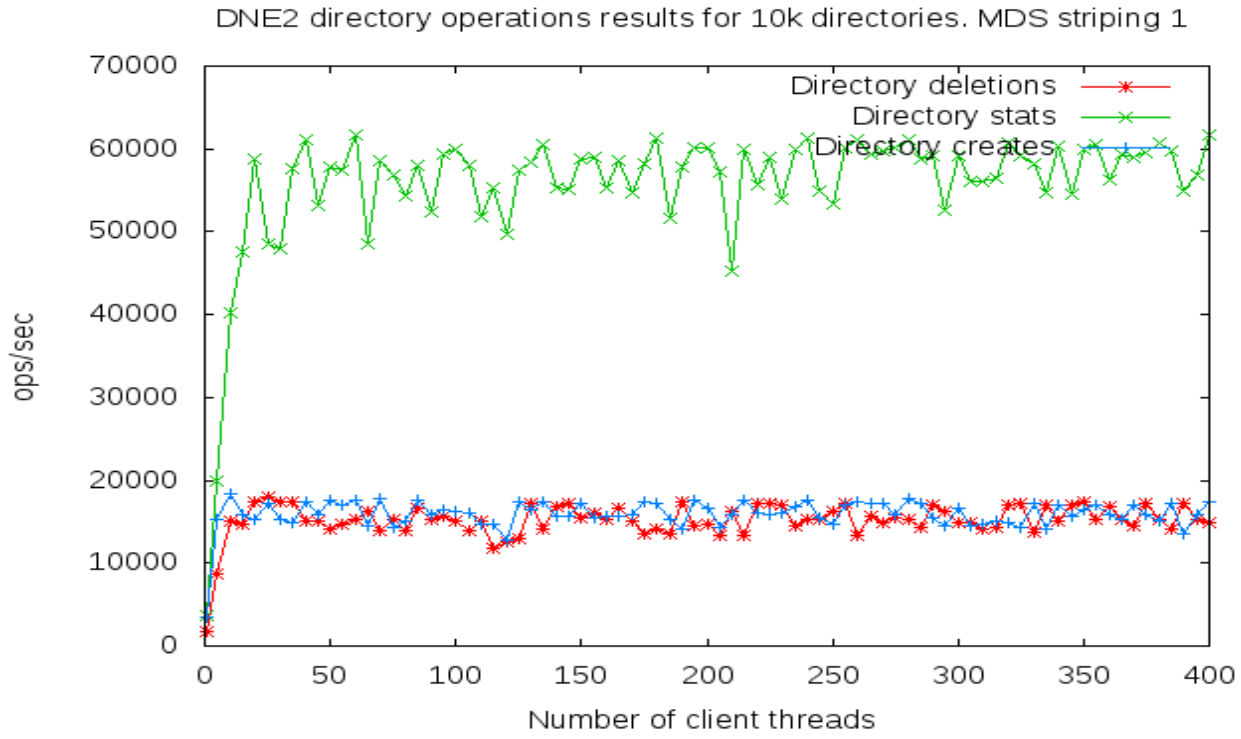
Figure 12: DNE P2 inheritance 100k files using 8 MDTs

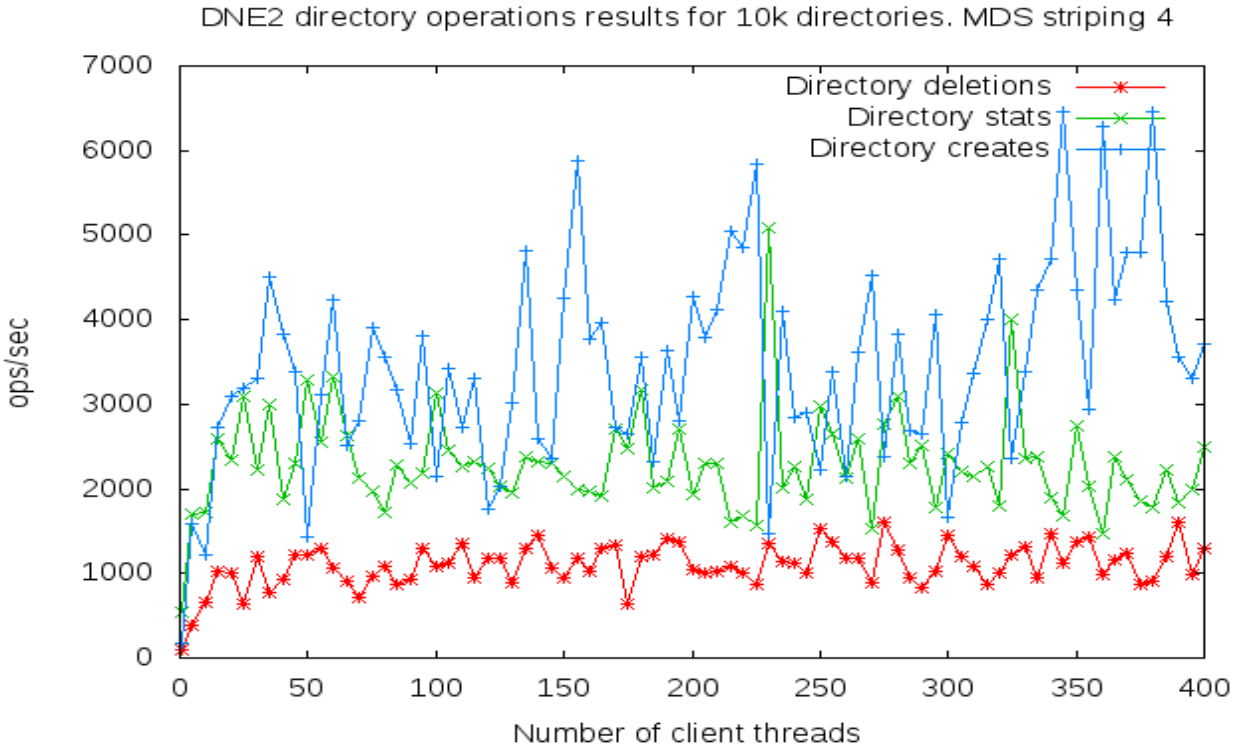
5.2 DIRECTORY PERFORMANCE FOR THE INHERTIANCE CONDITION

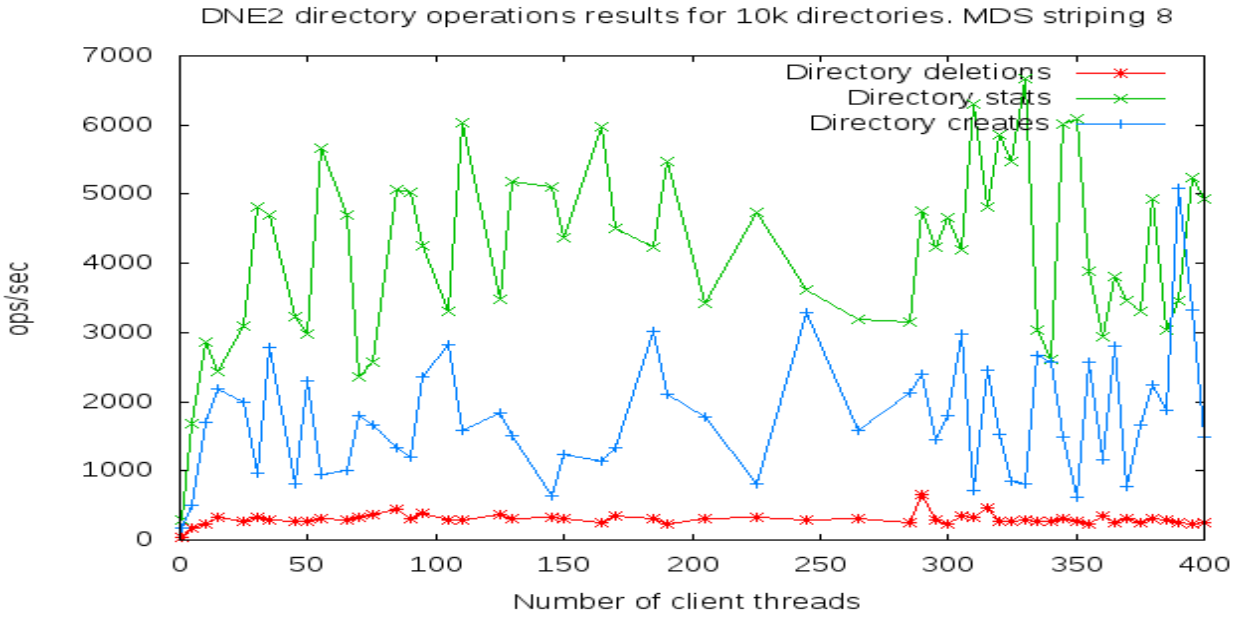
In the previous section for the case of subdirectories inheriting the striping patterns we saw a modest improvement in file creations and deletions. Here we will present the behavior of directory operations that have inherited the parent's DNE striping pattern. In the default case for a single MDS it can be observed what limitations are imposed on our user base. Both directory creations and deletions are the sever bottlenecks encountered by users. Typically, such operations are not routinely done during the application's life cycle. Directory stats have a reasonable performance. Much like the file operations case, the metadata server can be saturated around 35 nodes. This case is also observed as we scale the number of metadata servers the directories are striped across.

When scaling to higher striping across the MDS servers we observe a collapse in performance for every class of directory operations. As the striping count increases the performance degrades by large margins. Discussion with the Intel engineer behind this work revealed this is the expected case when MDS striping of directories are inherited to child subdirectories due to the over head needed to communicate the directories states between all the MDS servers. This state needs to consistent for recovery operations to operate correctly. The lesson learned here is that while we see a modest performance boost for file operations the directory operations suffer when the MDS striping patterns of a directory are inherited to their subdirectory children. The results are present to show to the reader how drastic the impact was. To make this complete the data results for the one hundred thousand directories are also presented to show at larger scales the problem still persist.

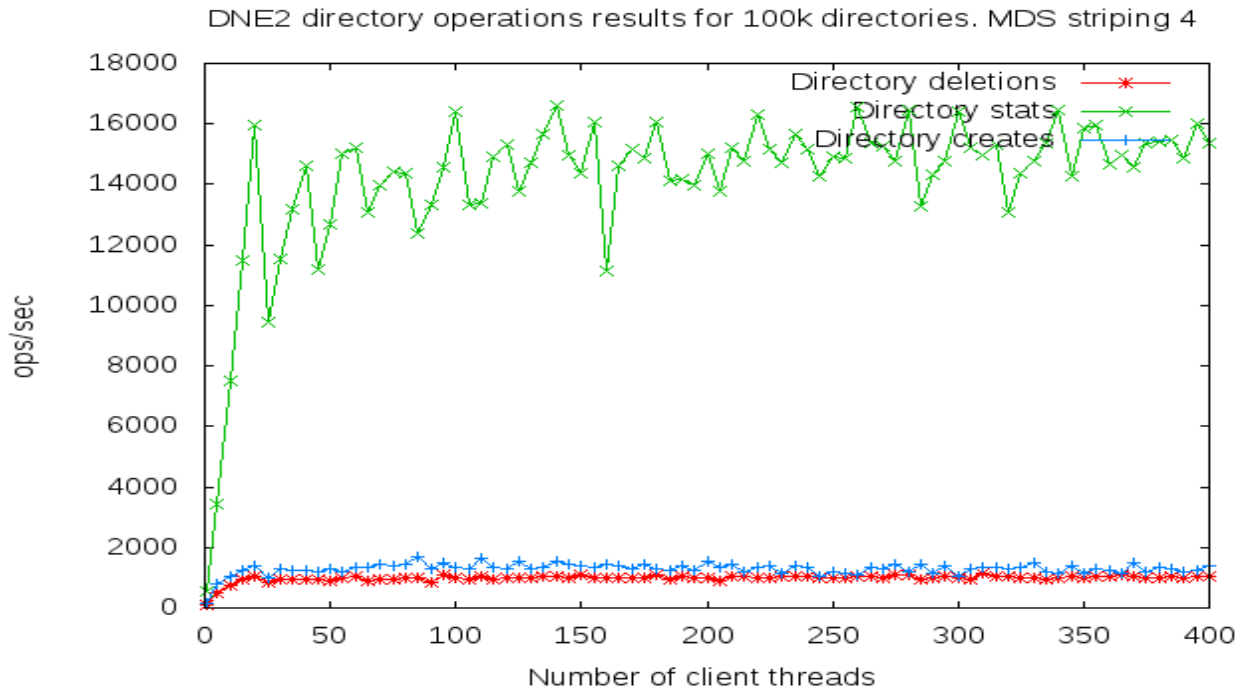
5.2.1 Ten thousand subdirectories performance




DNE2 directory operations results for 10k directories. MDS striping 4
Directory deletions
Directory stats
Directory creates
ops/sec
Number of client threads
0
1000
2000
3000
4000
5000
6000
7000
0
50
100
150
200
250
300
350
400


DNE2 directory operations results for 10k directories. MDS striping 8
Directory deletions
Directory stats
Directory creates
ops/sec
Number of client threads
0
1000
2000
3000
4000
5000
6000
7000
0
50
100
150
200
250
300
350
400

5.2.2 One hundred thousand directory results

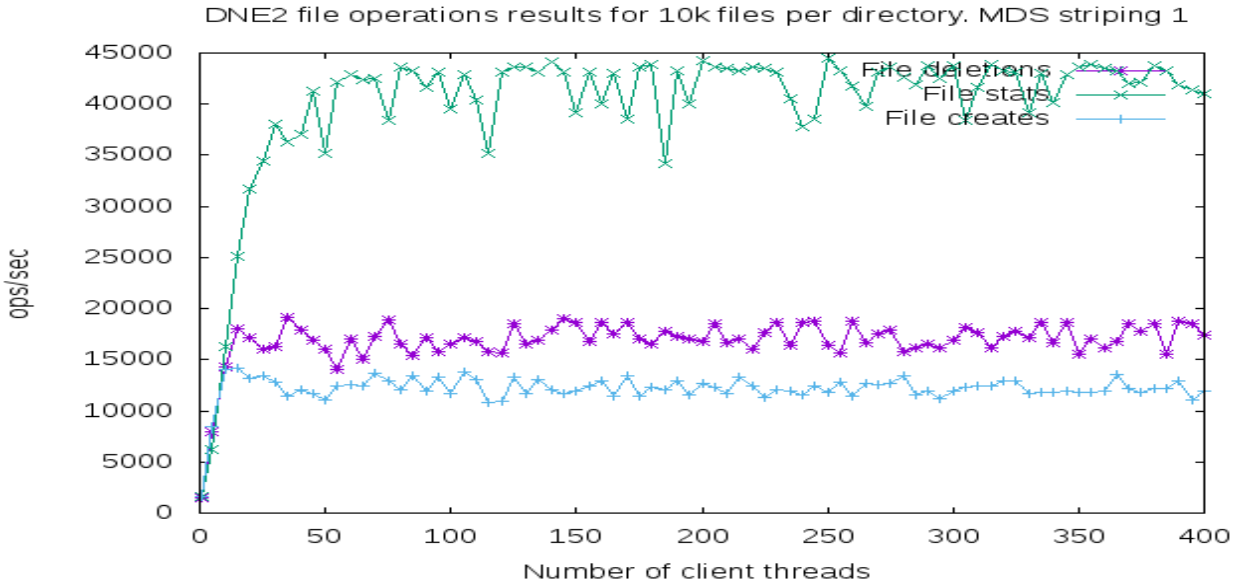


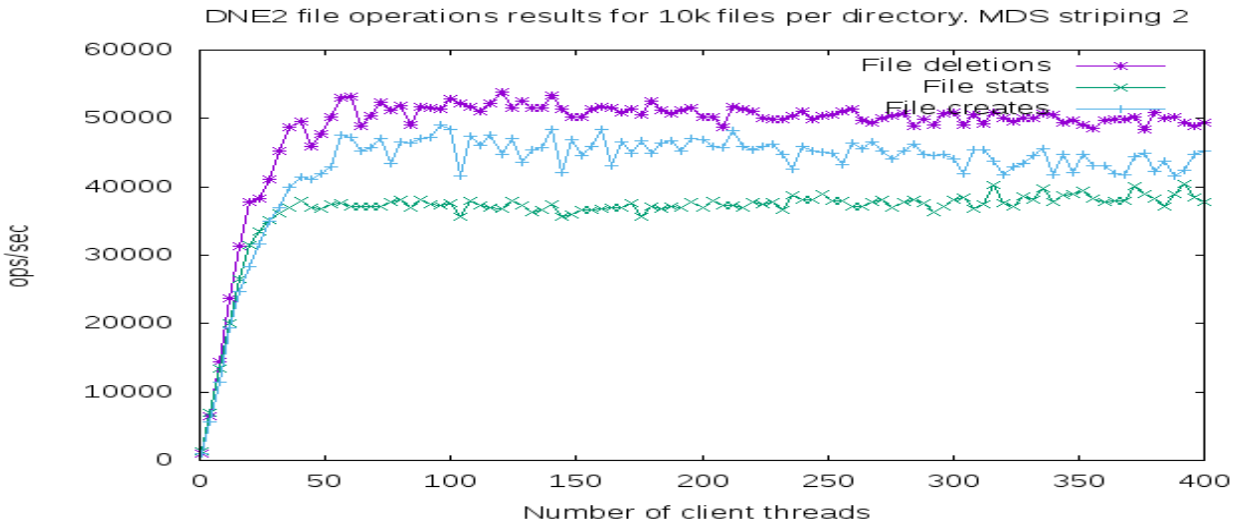
5.3 FILE PERFORMANCE FOR NON-INHERITED CASE

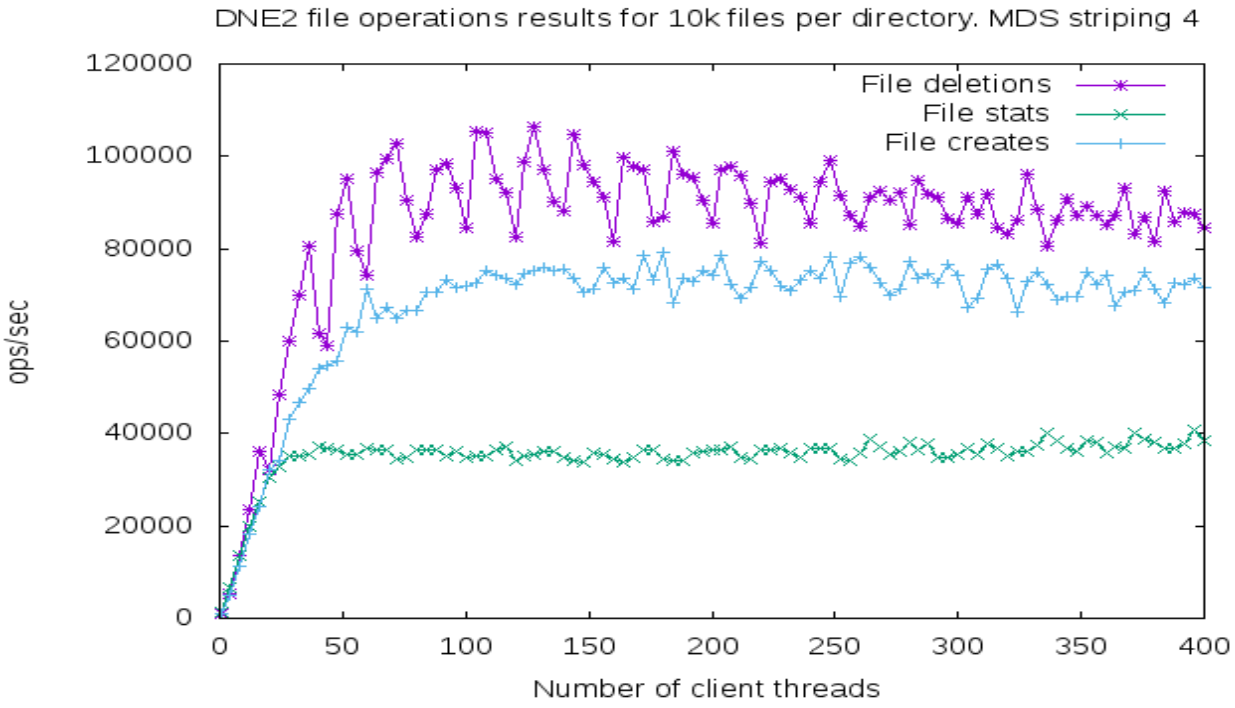
Examining the ten thousand files per directory shows excessive noise due to the small file sample size in each directory. With the latter results of larger file counts a clearer picture will merge for the results. For this setup, mdtest created unique subdirectories for each MPI process spawned on a client node. For our

test conditions each unique subdirectory was mapped to a specific metadata server. This allowed parallel MPI processes to perform the file operations in parallel on the client nodes against multiple metadata servers. This ensured maximum performance was reached.

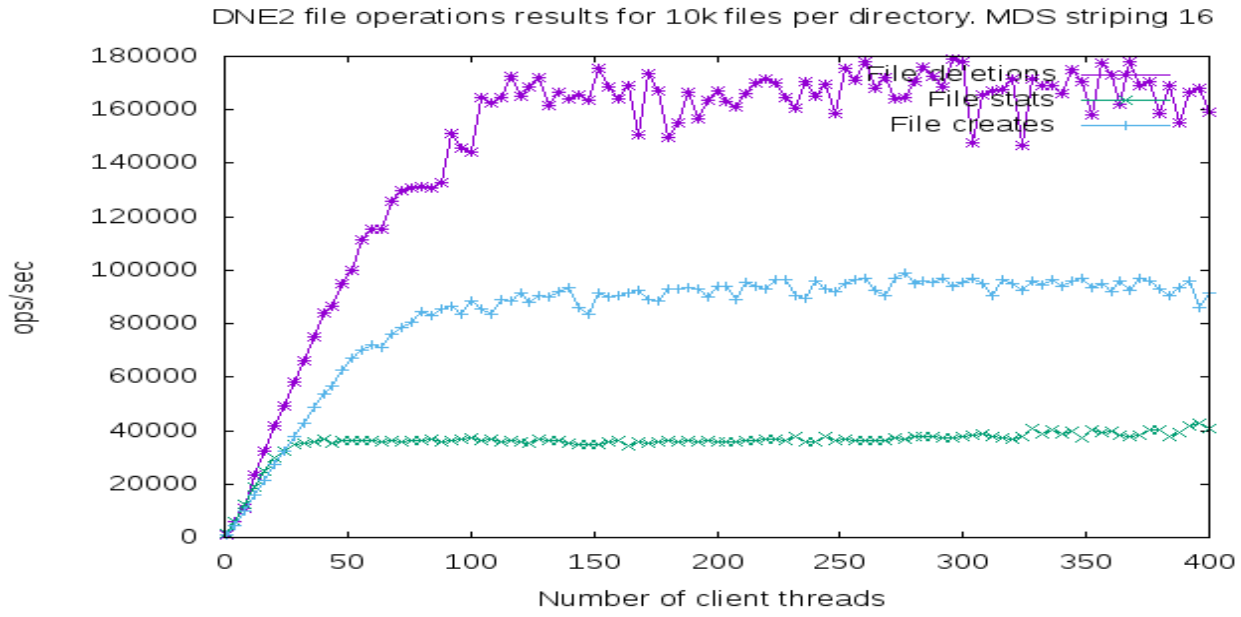
5.3.1 Ten thousand files per directory



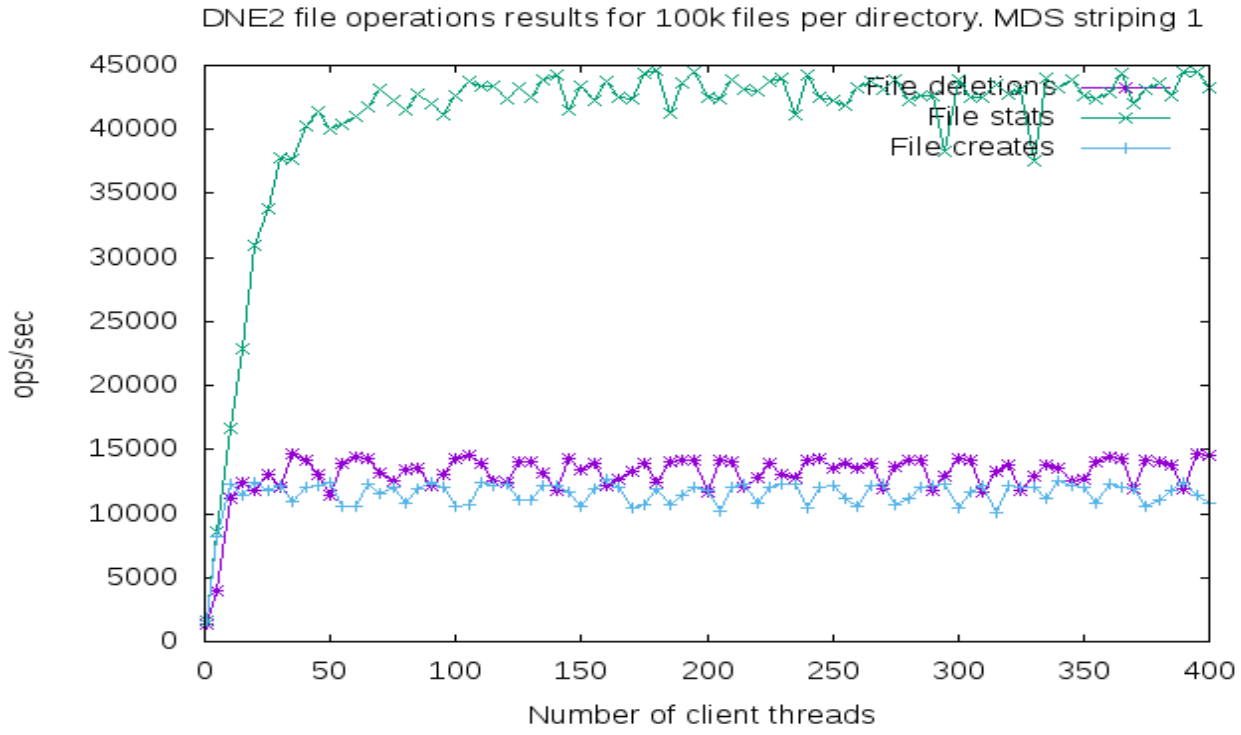



DNE2 file operations results for 10k files per directory. MDS striping 4
ops/sec
Number of client threads
File deletions
File stats
File creates
0
20000
40000
60000
80000
100000
120000
0
50
100
150
200
250
300
350
400

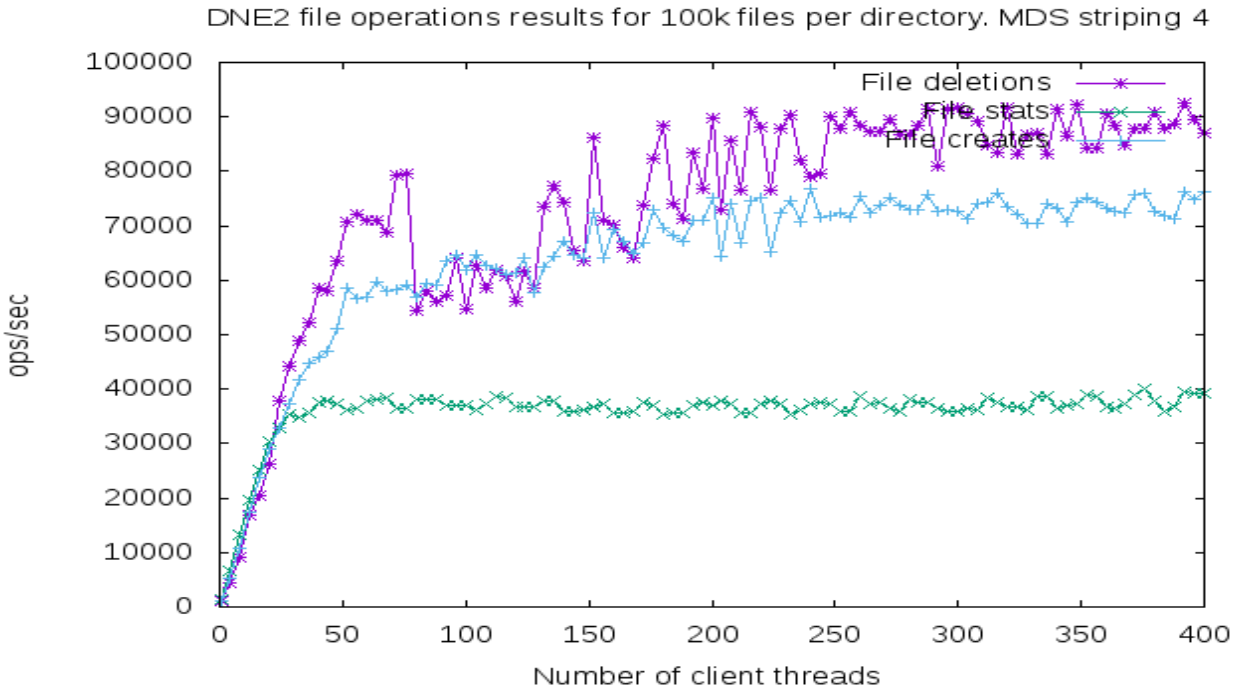

DNE2 file operations results for 10k files per directory. MDS striping 8
ops/sec
Number of client threads
File deletions
File stats
File creates
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
0
50
100
150
200
250
300
350
400

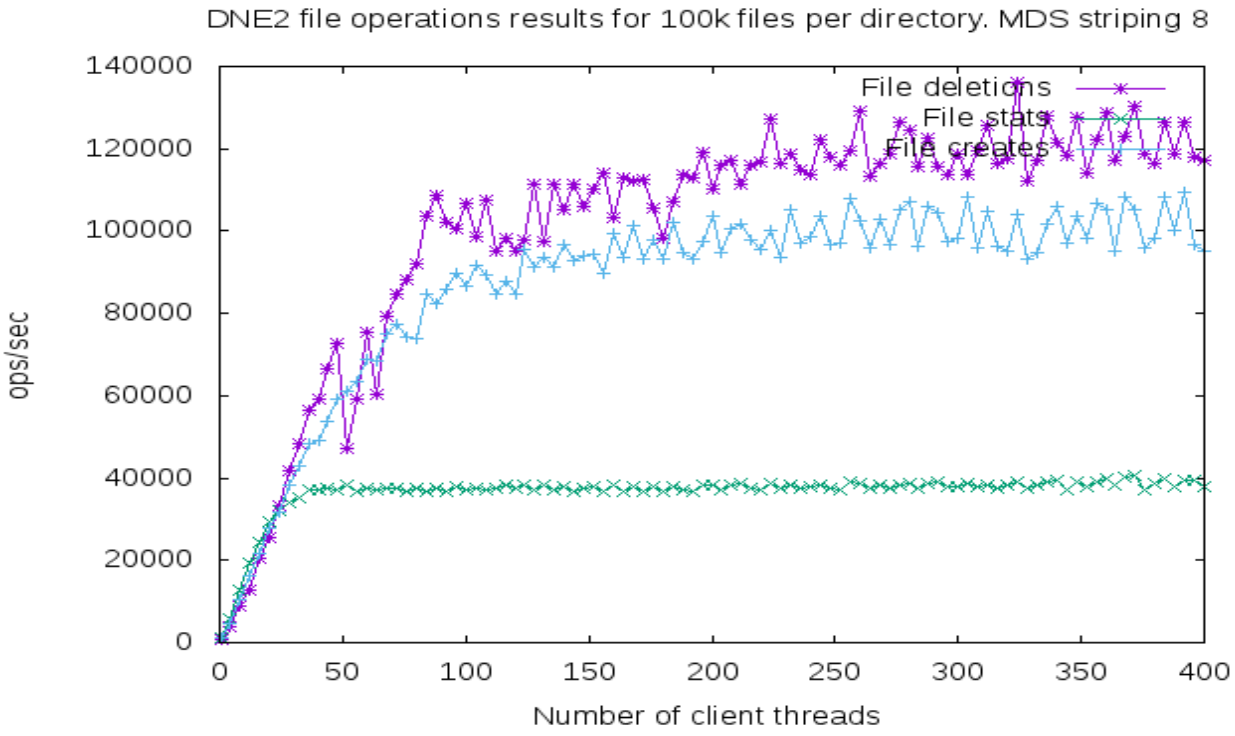


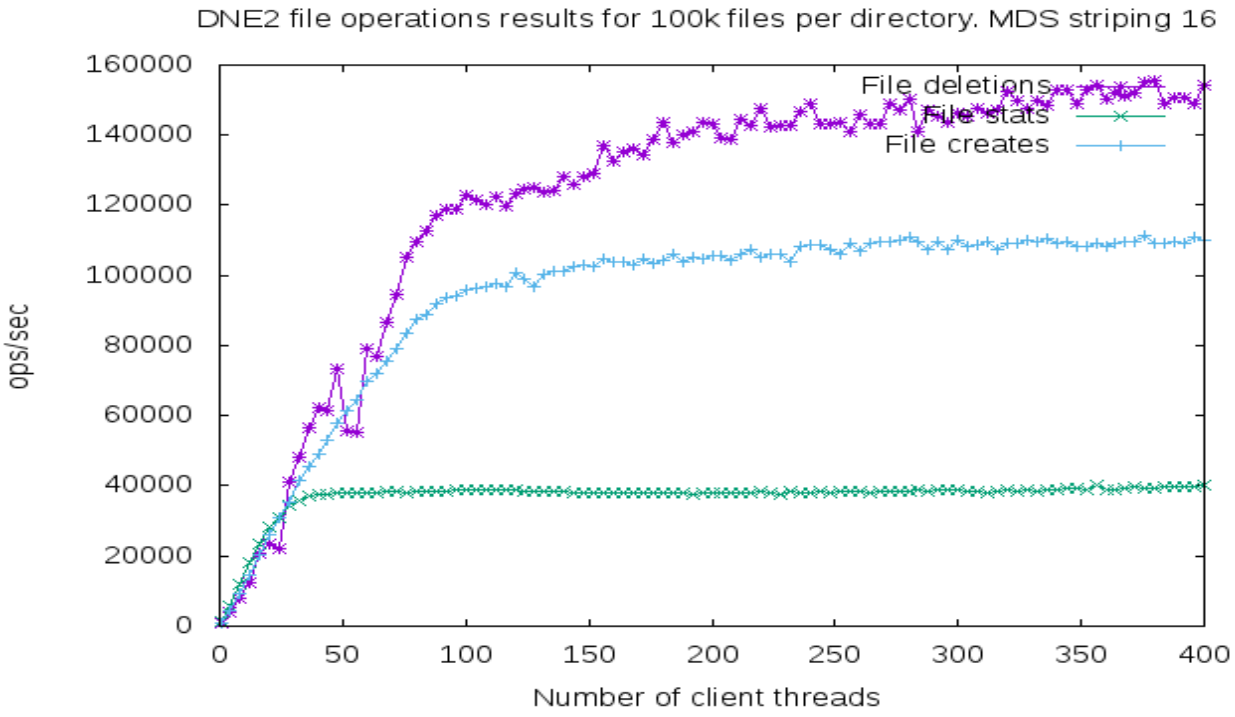
5.3.2 One hundred thousand files per directory




DNE2 file operations results for 100k files per directory. MDS striping 2
ops/sec
Number of client threads
File deletions
File stats
File creates
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
50
100
150
200
250
300
350
400


DNE2 file operations results for 100k files per directory. MDS striping 4
ops/sec
Number of client threads
File deletions
File stats
File creates
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0
50
100
150
200
250
300
350
400



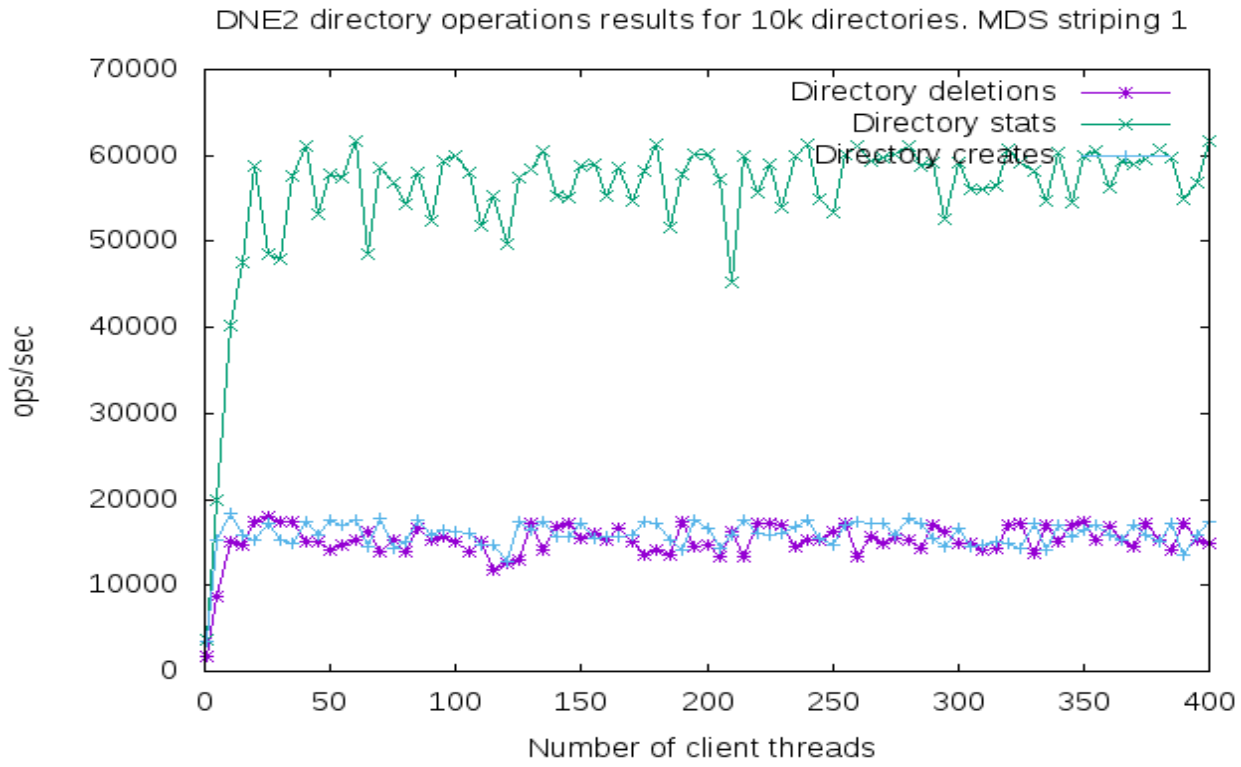


5.4 DIRECTORY PERFORMANCE FOR NON-INHERITED CASE

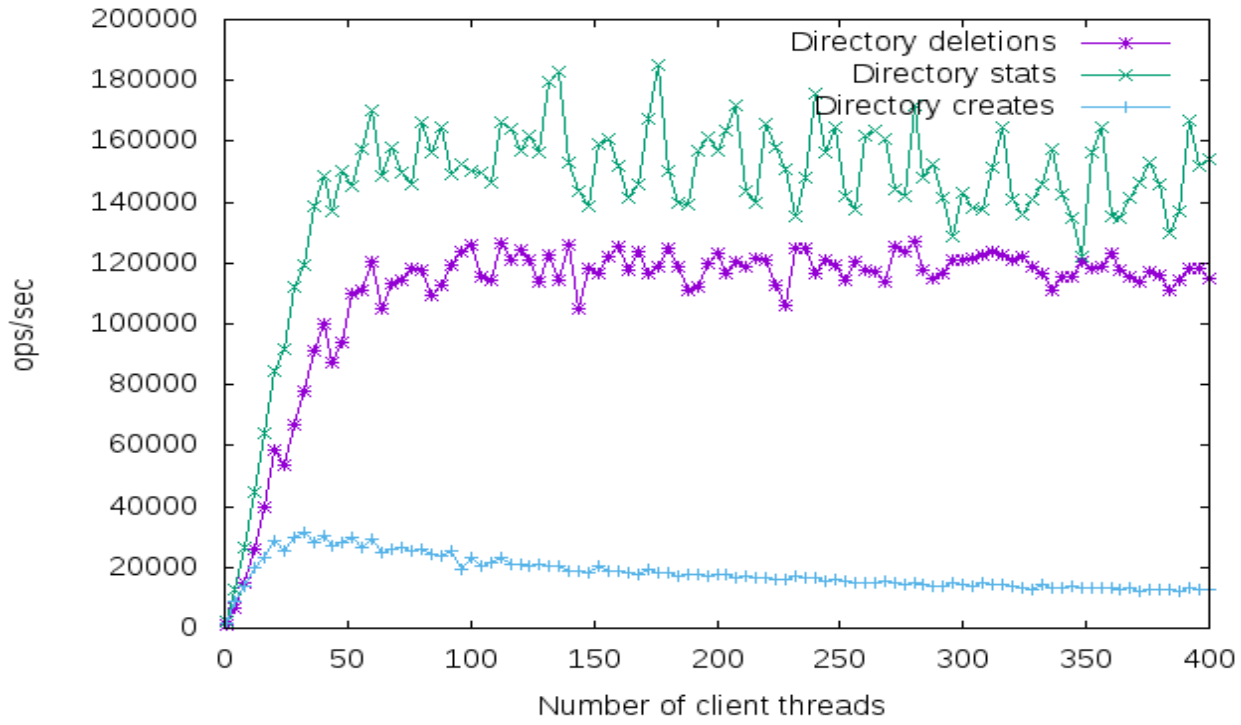
In the case of non-inheritance subdirectories are hashed across different metadata servers just as the files created in a DNE2 striped directory. Earlier we examined file operations behavior in the case where the files were located in these subdirectories. For this set of test the behavior of directory operations are

profiled for subdirectories that have been hashed across multiple metadata servers. The results demonstrate that the non-inheritance case for directory operations far outperforms the inherited condition. In all cases directory creation does not scale at any DNE2 striping setting. When compared to the inherited case this still far outperforms those results since for the inherited case subdirectory performance decreased with increasing DNE2 striping. For directory stats we see a doubling in performance going to two metadata servers. An increase to four metadata servers reaches the maximum possible performance with the test hardware setup. Increasing DNE2 striping using more than four metadata servers brought no additional benefit for the case of directory stats. For the case of directory deletions we do see a scaling in performance, though weaker scaling than stats exhibit. Like the directory stats, directory deletions also hit a performance ceiling when using DNE2 striping across four metadata servers. The performance also peaked at the same level as directory stats.

5.4.1 Ten thousand subdirectories




DNE2 directory operations results for 10k directories. MDS striping 2
Directory deletions
Directory stats
Directory creates
ops/sec
Number of client threads
0
20000
40000
60000
80000
100000
120000
140000
0
50
100
150
200
250
300
350
400

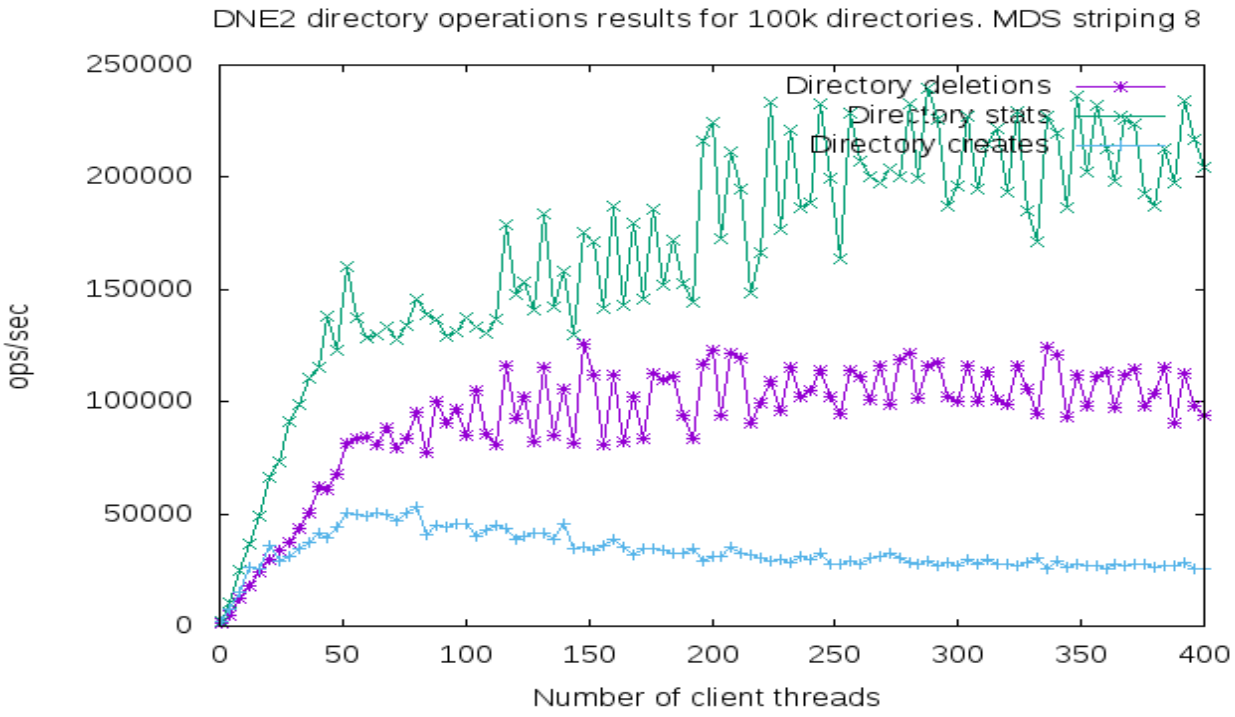

DNE2 directory operations results for 10k directories. MDS striping 4
Directory deletions
Directory stats
Directory creates
ops/sec
Number of client threads
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
200000
0
50
100
150
200
250
300
350
400


DNE2 directory operations results for 10k directories. MDS striping 8
ops/sec
Number of client threads
Directory deletions
Directory stats
Directory creates
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
200000
0
50
100
150
200
250
300
350
400


DNE2 directory operations results for 10k directories. MDS striping 16
ops/sec
Number of client threads
Directory deletions
Directory stats
Directory creates
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
200000
0
50
100
150
200
250
300
350
400

5.4.2 One hundred thousand subdirectories


DNE2 directory operations results for 100k directories. MDS striping 4
Directory deletions
Directory stats
Directory creates
ops/sec
Number of client threads
0
50000
100000
150000
200000
250000
0
50
100
150
200
250
300
350
400


DNE2 directory operations results for 100k directories. MDS striping 8
Directory deletions
Directory stats
Directory creates
ops/sec
Number of client threads
0
50000
100000
150000
200000
250000
0
50
100
150
200
250
300
350
400

6. PRODUCTION DEPLOYMENT

This report has focused mainly on the software side of the DNE Phase 2 rollout, however it is important to understand the OLCF's considerations for the hardware side of this upgrade. Each Atlas file system has a single metadata server per Atlas file system. The plan for production is this to increase to four metadata servers per Atlas file system, for a total of eight. Our current production metadata are dual-socket six-core Intel Sandy Bridge 2.5 GHz servers each with 256GB DDR3 RAM distributed evenly between the two processors, a single port FDR InfiniBand card, and a dual port 6Gb/s SAS card. Due to the limited abilities for SMP scaling in the Lustre MDS code and the added NUMA latency of having the IB card, SAS card, and Lustre MDS processes potentially living on different CPUs we decided to rethink our approach to building MDS servers.

The new systems will have a single socket eight-core Intel Haswell 3.2GHz processor, 256GB DDR4 RAM, a single port FDR InfiniBand card, and a dual port 12Gb/s SAS card. The new single-socket MDS nodes should reduce NUMA induced latency, the faster CPU frequency should increase responsiveness from the MDT threads (due to limited SMP scaling), and the faster SAS link to disk storage should also increase the performance for flushes (syncs) from MDS memory to the backend storage.

As part of the process of moving to new hardware, we are also making an investment in a new backend storage target for metadata. The two production systems in place currently share a NetApp E5500 storage array composed of 48 900GB SAS drives with a 6Gb/s interconnect to the MDS nodes. These will be replaced with two NetApp EF560 storage arrays each with 24x 1.6TB solid state drives (one storage array per file system), and 12Gb/sec SAS for host connectivity. Even though most of our metadata operations are in memory on the MDS nodes, we felt that having 8 metadata servers on a single NetApp E5500 could be a bottleneck due to the limited IOP throughput of SAS disks. Initial testing shows the NetApp EF560 all solid-state disk solution has provided over 30x improvement in IOPs at the block level.

6.1 HARDWARE EVALUATION FOR PRODUCTION DEPLOYMENT

We had to make many considerations in measuring the performance of the updated code. Without the use of a kernel I/O scheduler, Lustre metadata consists of 4 kB write sizes to the storage target. We do not

recommend running of a I/O scheduler. These small IO size lends itself to do very well with flash based storage because of the lower latency and higher IOPs.

Evaluation of the new storage system focused on tests at the block level and the Lustre level. Benchmarking the various features available on storage hardware is often a daunting task. In the case of the NetApp EF560, we tested read/write performance for all of the following scenarios:

- Varying I/O segment sizes (32 kB, 64 kB, 128 kB, 256 kB, and 512 kB)
- Enabling/disabling the DA (data assurance) feature
- Testing RAID 10 volume groups of varying number of disks (since some RAID subsystems do not handle large RAID volume group performance well)
- Using a parity decoupled disk volume instead of a RAID10
- Testing performance from a single node and from all 4 nodes attached to each disk system.

To test the block level IO performance we used a tool developed at the OLCF called *fair-lio*. The software creates a random 4 kB workload, and very effectively stresses out the disks system. Block-level tests are ongoing at the time of writing this document. It will be updated will new results in the near future.

After qualifying the performance of the block level performance we will evaluate the performance at the Lustre software layer. To do this we will create a small Lustre file system using the new metadata servers and new NetApp EF560 storage system as the metadata target. We will use the mdtest benchmark tool for these tests. File system level tests are not complete at the time of writing this document and it will be updated will new results in the near future.

7. CONCLUSIONS

The OLCF plans to greatly increase in the metadata performance over the coming months available to the Atlas file systems. This will be through the deployment of new metadata server hardware and distributed namespaces within Lustre. From testing, four metadata servers per file system appear to offer the most value for hardware without potential detriment to performance. Due to the overhead of metadata striping inheritance, OLCF must carefully consider how to implement DNE. One potential design choice would be to stripe top-level project and user directories across the new metadata targets. Additionally, since existing directories cannot have their metadata restriped, this feature will target new users and projects.

8. ACKNOWLEDGMENTS

This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

APPENDIX A – MDTEST SCALING TEST SCRIPT

```
#!/bin/bash
#PBS -l nodes=20
#PBS -l walltime=24:00:00
#PBS -N results-mdtest-scale
#PBS -j oe

MOUNT=<filsystem-name>
OSTCOUNT=$(lctl get_param -n lov.$MOUNT-clilov*.numobd)
ITER=5

PBS_JOBID="dne2_8_mds"
BINDIR=/lustre/$MOUNT/$USER
OUTDIR=$BINDIR/${PBS_JOBID}_md_test
[ -e $OUTDIR ] || {
        mkdir -p $OUTDIR
        lfs setstripe -c $OSTCOUNT $OUTDIR
}
cd $BINDIR

#for filecount in 10000 100000 1000000 10000000
for filecount in 1000000 10000000
do
        TOTAL=$(($filecount / 1000))

        for threads in 1 $(seq 5 5 400)
        do
                nodes=$(((threads + 19) / 20))
                [ $nodes -gt 20 ] && nodes=20

                aprun -n $threads -N $nodes $BINDIR/mdtest -I $(( $filecount / $threads)) -i $ITER -d
$OUTDIR/shared_${TOTAL}k_$threads
        done

        #for threads in 1 $(seq 5 5 400)
        #do
        #       nodes=$(((threads + 19) / 20))
        #       [ $nodes -gt 20 ] && nodes=20

        #       aprun -n $threads -N $nodes $BINDIR/mdtest -I $(( $filecount / $threads)) -i $ITER -u  -d
$OUTDIR/unique_${TOTAL}k_$threads
        #done
done
```