Lustre filesystem performance benchmark on build workload

# Architecture

There are many ways to configure Lustre filesystem depending on the characteristics of the workload patterns. The following configurations have been tried for this study. More specifically, two features related to small file performance (<http://193.62.125.70/CIUK-2016/AdamRoe.pdf>) have been tried (Configuration 2 and Configuration 3):

* DNE Phase II – Allow metadata server and storage target to scale horizontally, available from Lustre 2.8+ (<http://wiki.lustre.org/Lustre_Metadata_Service_(MDS)>)
* Data-on-MDT – Allow small data to be written directly on metadata server, available from Lustre 2.11+ (<http://wiki.lustre.org/Data_on_MDT_Solution_Architecture>)

No data striping is enabled for files, because 99.7% of the files from our test workload (GCC4.9.2 build) are smaller than 1MB in size (following Lustre best practice guidance).

No cluster-wide flock support is enabled. This is not needed for our distributed build application, as there will be no concurrent access of the same file from multiple hosts.

Configuration 1:

Single metadata server (MDS) backed by one metadata target (MDT), four object storage servers (OSS) each backed by an object storage target (OST). All servers use AWS m5.xlarge instance (4 CPUs, 16GB mem, up to 10Gb network). Both MDT and OST are general purpose SSD (maximum burst of 3000 IOPS for 30 minutes) formatted with LDISKFS (a variant of EXT4)

gp2 SSD

LDISKFS

MGT/MDT0

OST0

OST1

OST2

OST3





MGS/MDS

m5.xlarge

OSS

m5.xlarge

OSS

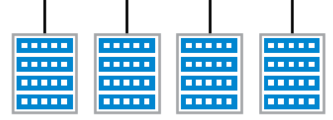
m5.xlarge

OSS

m5.xlarge

OSS

m5.xlarge



10GbE

Lustre Clients (1 – 40)

m5.xlarge (4 CPUs, 16 GB Mem)

Configuration 2:

DNE II. Two or four MDS each backed by an MDT, four OSS each backed by an OST. All MDT and OST are general purpose SSD formatted with LDISKFS. File system directory metadata is distributed among all MDS with DNE II feature (Distributed namespace with striped directories).

OST0

OST1

OST2

OST3

DNE MDT0 – MDT3

MGT/MDT0



…



MGS/MDS

m5.xlarge

MDS

m5.xlarge

MDS

m5.xlarge

OSS

m5.xlarge

OSS

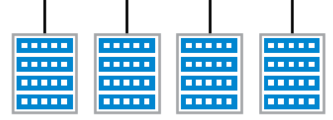
m5.xlarge

OSS

m5.xlarge

OSS

m5.xlarge



10GbE

Lustre Clients (1 - 40)

m5.xlarge (4 CPUs, 16 GB Mem)

Configuration 3:

Data-on-MDT. Four MDS each backed by an MDT, four OSS each backed by an OST. All MDT and OST are general purpose SSD formatted with LDISKFS. File system directory metadata is distributed among four MDS. The first 128KB of each file is configured to be placed on MDT. Otherwise, the architecture is the same as the one in configuration 2.

## Software

|  |  |
| --- | --- |
| Client machine | CentOS 7.4.1708  3.10.0-693.21.1.el7.x86\_64  **Lustre Client 2.11.0**  The following tuning has been done on all client machines to improve small file performance  **Enable client-local flock support when mounting the Lustre client. This is possible because the distributed build application will not access the same file from multiple hosts.**  *mount -t lustre -o localflock …*  **Disable network checksum.**  *lctl set\_param osc./\*.checksums=0*  **Increase maximum RPCs in flight from 8 to 32.**  *lctl set\_param osc./\*.max\_rpcs\_in\_flight=32*  **Increase maximum client side dirty page size from 32MB to 128MB.**  *lctl set\_param osc./\*.max\_dirty\_mb=128* |
| Server machines | Metadata Server:  Each node has:  CentOS 7.4.1708  3.10.0-693.21.1.el7.x86\_64  **Lustre 2.11.0** |

# Application

We use the same benchmarking application that was used in the GlusterFS/NFS-Ganesha vs NFS benchmarking. The following table summarizes the key characteristics.

The main I/O access patterns are sequential reads (source files) and random writes (objects, archives, and binaries). The build application benefit hugely from client-side caching, which is supported by Lustre client.

|  |  |
| --- | --- |
| Workload  Characteristics | GCC4.9.2 build |
| Total number of files in the source tree | 84895 |
| Total storage space before build | 528MB |
| Total number of build tasks | 7851 |
| Average run time of each task | 0.43 seconds |
| Largest file generated after build | 225MB |
| Percentage of files smaller than 1MB after build | 99.7% |
| Percentage of files smaller than 64KB after build | 97% |
| Total storage space after build | 3.2GB |
| Level of sub-makes | 6 |

# Test Results

## Use Case 1: Copy

This use case tests the effect of multiple clients copy an entire GCC source code directory (528MB, 84895 files) from a local directory to the Lustre client mount at the same time under three different configurations. The data for 30 clients and 40 clients concurrent copy with single MDS is not complete, so we are not adding that to the comparison.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Configurations  concurrent clients | Configuration 1:  Single MDS | | Configuration 2:  Multiple MDS with DNE.  **No Data-on-MDT**. | | | | Configuration 3:  Multiple MDS with DNE and **Data-on-MDT** | |
| Average copy time in seconds | Max IOPS used | Average copy time in seconds | | Max IOPS used | | Average copy time in seconds | Max IOPS used |
| 2 MDS | 4 MDS | 2 MDS | 4 MDS | 4 MDS | 4 MDS |
| 1 | 105 | 300 | **134** | **151** | 286 | 644 | **135** | 450 |
| 10 | 135.3 | 2123 | **173.4** | **173.2** | 3900 | 4004 | **196.8** | 3700 |
| 20 | 155.1 | 5469 | **236.2** | **248.7** | 8602 | 9692 | **288.2** | 8018 |
| 30 | - | - | **354.6** | **353.3** | 11289 | 12449 | **416.1** | 9172 |
| 40 | - | - | **415.3** | **437.7** | 11700 | 11545 | **527.4** | 11400 |

Observation:

* Lustre data on metadata (DOM) feature does not bring benefit to the copy use case when the number of concurrent clients increase (see the green bar). In fact, it slows down the copy (20% increase in copy time on average)
* The number of metadata servers does not impact the performance of the copy very much (see the orange and blue bars)

We then compare the average copy time from local disk to different file system mount, as shown in the following graph.

Observation:

* Lustre file system performs the best out of the three file systems with regard to the copy use case. This clearly shows the benefit of the scalability of the Lustre file system.
* Lustre file system resource usage is much slower than that of the GlusterFS/Multi-head NFS ganesha solution.

## Use Case 2: Concurrent Build

This use case tests the performance of multiple clients doing GCC 4.9.2 build in the Lustre mount at the same time.

The build command executed by each client is “make -j8”. It is observed that during the build, all client hosts can utilize 100% CPU.

Note that with concurrent build use case, each build is done on one client host, that is, all build tasks from a build are all run on the same host. This is very different than distributed build use case, where build tasks from a build can land on many different hosts in a build cluster, potentially causing many extra RPC calls from Lustre client to MDS because of cache misses.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Configurations  concurrent clients | Configuration 1:  Single MDS | Configuration 2:  Multiple MDS with DNE.  No Data-on-MDT. | | Configuration 3:  Multiple MDS with DNE and Data-on-MDT |
| Average build time (seconds) | Average build time  (seconds) | | Average build time |
| 2 MDS | 4 MDS | 4 MDS |
| 1 | 1910 | 1504 | 1503 | 1487 |
| 10 | 1931 | 1480 | 1468.7 | 1494 |
| 20 | 2165.6 | 1558.1 | 1549.6 | 1520.6 |
| 30 | - | 1586.8 | 1582.1 | 1611.1 |
| 40 | - | 1691.5 | 1730.5 | 1706.7 |

Observation:

* Multiple metadata servers bring significant performance improvement to the concurrent build use case. Even with only one client, the average build time is reduced from 1910 seconds (almost 32 minutes) down to 1500 seconds (25 minutes).
* Lustre data on metadata (DOM) feature does not show noticeable benefit with concurrent build use case.
* The number of metadata servers does not impact the performance of the concurrent build use case.
* The system scales well. The average build time only increases by 13% from one concurrent build to 40 concurrent builds.

We then compare the average copy time from local disk to different file system mount, as shown in the following graph.

GlusterFS servers using **m5.2xlarge** to handle more clients

GlusterFS data not collected

GlusterFS servers using **m5.xlarge** resource usage close to max out

Observation:

* Overall Lustre file system performs the best out of the three file systems regarding the concurrent build use case. This clearly shows the benefit of the scalability of the Lustre file system.
* Lustre file system resource usage is much slower than that of the GlusterFS/Multi-head NFS ganesha solution.
* Lustre file system behaves very stable, with no data corruption after many rounds of tests including copy, delete and compile. GlusterFS/NFS-Ganesha combo, on the other hand, has 20% – 30% of the concurrent builds failure due to data corruption.

## Use Case 3: Distributed Build with Elastic Make

The following diagram shows the architecture of the distributed build use case using elastic make. With distributed build, tasks from one build (e.g., ar, link, etc) can land on any build host with no cached data, resulting in increased RPC calls to retrieve metadata information and to get the actual data. This imposes higher demand on the file servers

OST0

OST1

OST2

OST3

DNE MDT0 – MDT3

MGT/MDT0



…



MGS/MDS

m5.xlarge

MDS

m5.xlarge

MDS

m5.xlarge

OSS

m5.xlarge

OSS

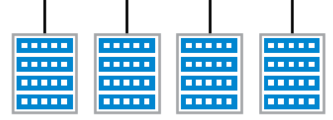
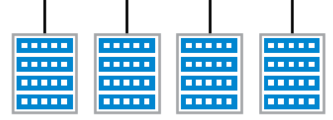
m5.xlarge

OSS

m5.xlarge

OSS

m5.xlarge



10GbE

20 build server hosts

m5.xlarge (4 CPUs, 16 GB Mem)

1 slot per CPU, 80 slots in total

mount /mnt/luster with Luster client

1 - 20 elmake clients

m5.large (2 CPUs, 8 GB Mem)

mount /mnt/luster with Luster client

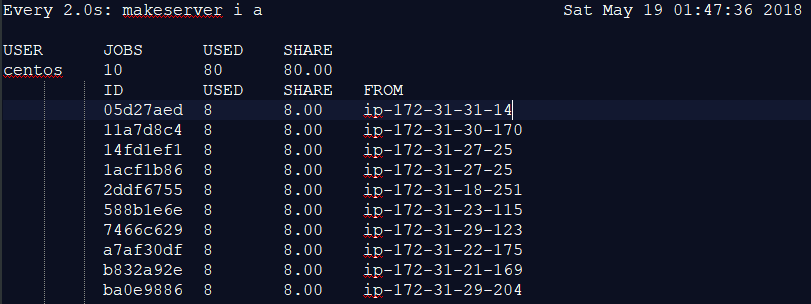
|  |  |  |
| --- | --- | --- |
| Number of clients | Single MDS  Average build time (seconds) | 4 MDS with DNE.  No Data-on-MDT  Average build time (seconds) |
| 1 | 825 | 614 |
| 10 | - | 1284 |
| 20 | - | 2172\* |

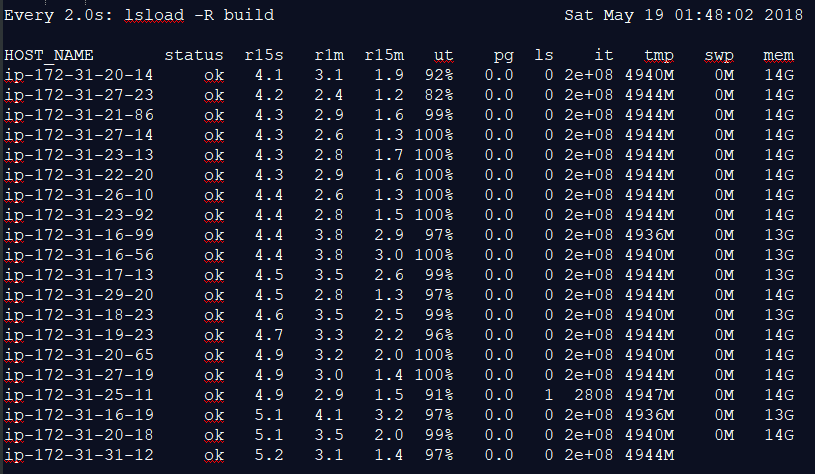
\* 1 client build failure (root cause not clear, seems to be object file data corruption)

Observation:

* Overall Lustre performs as expected. When all build slots are used, CPU ut on build hosts can be maximized. In the meantime, the resource usage on Lustre servers are minimal.

When one client builds GCC 4.9.2 using elmake utilizing all 80 slots in the build cluster, the build is finished in 10 minutes 14 seconds. When 10 clients build at the same time, the average build time is 21 minutes 24 seconds.

The following is a snapshot of the system when 10 clients are doing elmake build of GCC 4.9.2 at the same time: 



* When 20 clients are doing elmake at the same time, the average build time is 36 minutes 12 seconds. However, the build fails for one client, and there is no way to recover without running ‘make clean’. It seems that one object file cannot be recognized, indicating data corruption. The root cause is not clear.
* During one round of test, the Lustre mount on all clients seems hang. I do not have the required knowledge to find out the root cause. I don’t know if it is an environment issue, or a bug in Lustre. Restarting all the Lustre servers and wait for the system to recover (takes 15 minutes) resolve the issue.

# Summary

* Configurations that target small file optimizations have been tried when setting up the Lustre file system:
* DNE Phase II – Distributed namespace with striped directories, allowing metadata server and storage target to scale horizontally.
* Data-on-MDT (DOM) – Allow small data to be written directly on metadata server.

The best configuration for our build use case is DNE Phase II without data-on-MDT enabled. This configuration provides the best performance and the simplest configuration steps.

* Under the above configuration, concurrent builds in Lustre mount has the best overall performance compared to GlusterFS/Multi-head NFS-Ganesha and kernel NFS.
* Lustre file system is implemented as kernel module and consumes much less resource than GlusterFS/NFS-Ganesha under the same workload.
* Lustre file system runs very stable and reliable under the concurrent build workload. There is not a single case of data corruption observed during concurrent build benchmarks, where many rounds of copying/deleting/compiling from up to 40 clients are performed.
* When benchmarking the distributed build workload (with elastic make), the following problems are observed:
* There is one case of (potential) data corruption observed when 20 clients are building at the same time.
* There is one case of lustre mount hangs on all clients. Restarting the Luster server and wait for the system to recover seems to resolve the problem.
* All benchmarking results are obtained when the backend storage targets are formatted with LDISKFS. Formatting the storage targets with ZFS are not tried, so the performance penalty of ZFS is not clear.
* The overall impression is that when Lustre is tuned specific for the application, it does provide good performance (compared to GlusterFS) and scales well (compared to kernel NFS). However, if there are any issues with data corruption or file system hangs, it will be very hard to debug.