Lustre Modification

Design Problem and Challenges

This module is a modification to Lustre to address issues that exist with the file system's metadata server (MDS). Currently the file system provides only a standby architecture for the MDS. The current approach exposes the system to a single point of failure as it only uses one shared disk for its two MDSs. The intent of the module is to implement a process that replicates the MDS on several nodes, thus providing a higher degree of MDS availability. Note that baseline Lustre provides no replication capability, which is a severe weakness in its architectural design.

Some direction to how this architectural change might be effected can be found by borrowing from the symmetric active/active architecture for the MDS of PVFS which provides some design guidance for a similar type issue. In that instance an internal replication of the MDS was implemented with the use of Transis as a group communication facility. In Lustre one can get similar results by isolating the MDS from the other system components and then replicating the MDS. The MDS replication can be done either externally or internally. A detailed discussion of the pros and cons of each is available at the Lustre wiki.

For a number of reasons we chose to replicate externally. This approach constructs the group communication system like a wrapper around the MDS. The group communication system is placed into the Client-MDS communication path as an intermediate communication process. This process intercepts the network calls to and from the MDS and distributes the TCP packages to the group communication facility. The obvious benefit of this approach is that there is no need to touch the MDS code. Conversely, this approach will more than likely result in higher latency time due to the added overhead of inter-process communication. Additionally, the exact communication protocol and format between the MDS and the client must be known. This is an issue since the Lustre code manuals and wiki are uncharacteristically silent on this issue.

Finally, this modification requires a slight reconfiguration of Lustre from the baseline configuration approach. In this modified configuration, Lustre and the network components are configured in a single XML file. In this way the file system assumes that every node in the file system uses the same XML file for the configuration and startup. However, there appears to be no significant issue in utilizing different XML files for separate nodes. A detailed discussion of the reconfiguration process is not provided here due to space limitations.

Another critical issue in this redesign that needs to be addressed is the single instance execution problem. Since MDS members operate in synchrony each MDS has to generate the same output. The group communication software needs to be

designed to ensure that correct output is sent once and only once to any requesting component in the file system.

A general problem in any Lustre modification relates to source code analysis. Any such effort is highly labor-intensive since almost no comments exist in the baseline code. Additionally, the Lustre design is very complex and complicated making the code fairly difficult to read and understand. For example, Lustre runs nearly all components as kernel modules and publishes most of the functions to the kernel namespace. In this way, components can be called at any time from any point in the kernel. This issue makes it difficult to identify the function call path as a developer would in a normal program.

Implementation challenges were also encountered in implementing the modification, the most significant discussed below as follows:

- S**ystem configuration is fairly inflexible**

  Lustre wants the setup defined in advance. A script is then generated, thus configuring the entire file system. From this script a XML file is created. The XML file starts Lustre. Due to the Lustre security model, only messages from nodes configured in the XML file are valid. The problem is that in a normal Lustre configuration all messages from the interceptors are rejected. Hence, to get Lustre working with interceptors, the file system must be reconfigured.

- **Lack of message routing**

  Because of Lustre's security model, the routing of messages is not allowed. Messages not sent directly are discarded. In order to route messages the redesign essentially needs to fool the file system. Messages are changed in a manner whereby Lustre believes that the messages were sent directly.

- **Distributed locking mechanisms**

  Lustre uses only one MDS to serve thousands of clients. In order to keep the metadata state of the file system consistent, Lustre uses distributed locking mechanisms. Such mechanisms can create problems in the setup of an MDS group. The workaround for this issue is too detailed for the scope of this paper, but the solution uses a dynamic group reconfiguration approach.

- **Existing standby failover behavior of MDS**

  Lustre provides a standby high availability solution. It is possible to shut down the running MDS and to then start the backup MDS. The shutdown allows a commit all pending requests to disk. The problem is that only one MDS can be running at a given time. In baseline mode it is not possible to start the backup MDS as long as the active MDS is still running. The other problem is that only two MDS can be configured. These limitations make a setup of the

MDS group difficult. To run a proper MDS group actively, the system needs to start and run two or more MDSs at the same time. These issues also keep the dynamic group reconfiguration from properly functioning consistently.

Performance Testing

The testing of the above described modifications to the baseline Lustre code was performed on the same system as described in the experimental evaluation section of the paper. Initially, a system test was performed to verify that the functionality described by the modification was working as expected. Then, a series of performance tests were carried out. The results are summarized below.

For these performance tests the file system cache was deactivated. This allowed a comparison of the different test conditions. Tests were conducted using network setups of 100mb and 1GBit (only the 1G table is presented below in the interests of brevity). To evaluate the performance a benchmark program was written. The program created a set number of files, read the metadata of the files, and, at completion of the run, deleted the files. In order to evaluate system performance, the program ascertains the time needed for each operation. In order to obtain a mean measurement across trials, the program performs a number of test runs and then computes mean time for each operation across test runs.

Table: High Availability Lustre Redesign – 1G Bit Test for 100 files

Time per Op (msec)	create	read	delete
Standard Lustre	1.586	1.912	0.511
MDS Interceptor	120.203	79.432	38.995
Client Interceptor	121.363	83.731	40.004
Client + MDS	121.692	82.183	40.102
Redesign	122.583	122.163	40.204

The results of the test runs are shown in the table above. The default Lustre setup performs up to 75 times faster than the tested redesign. This performance impact is somewhat surprising. Other projects which used internal replication achieved

latency times of about 200ms; however these times result from external replication design and implementations. The measured latency times in the test runs are in the range from 185ms - 35ms, depending on the operation and network type. These results seem plausible, but the issue is the overhead applied to the file system. The measured overhead here is significantly higher in the redesign as compared to the baseline Lustre implementation or compared to previous projects implemented using internal replication.

The lessons from prior work in this area demonstrate that high availability doesn’t come without tradeoffs, but generally the performance impact is reasonable and the advantage of higher availability compensates for this negative. This is not the case here. The observed latency times introduced by the redesign are significantly high that they cannot be ignored. Hence, a review of the possible reasons for these high latency times is warranted.

The first step was to measure the delay times caused by the network. This was done with a test program. This program sent byte packages of increasing size over the given network path to measures the latency time caused by the network and calculate the bandwidth of the connection. Given the size of metadata messages, the test runs demonstrated that the latency time of the network was in the range of milliseconds. Average metadata messages of Lustre are not bigger than 1KB. For the Gigabit network test, this latency time even for the longest path was not much more than 200µs. So the network is not likely to be cause of the performance issues observed in implementing the redesign.

Another explanation is the implementation of the redesign itself. The core redesign change was in the message routing of the Lustre system. The proper functionality of this module was confirmed in system testing. The parallel approach is a bit faster than the serial used in the performance tests. However, the incremental performance was so little as to be negligible. As a result, the redesigned implementation demonstrated no errors responsible for causing the significant performance impact witnessed in performance testing.

The final explanation for the performance problems observed resides in Lustre itself. The file system cache was deactivated in order to achieve consistent results during performance testing. But due to the complex nature of the Lustre design, it is likely that Lustre uses internal techniques that are blocked by the interceptors and are the source of the performance impact that we observed in implementing and testing the modification. However, this is still an unproven hypothesis.

Aside from the performance issues encountered, it is worth examining the measurements from this modification study. In general, the trend of the values is acceptable. One can see that the test runs performed on 1Gbit network give lower latency times/more operations per second than the test runs performed on 100MBit network (not shown). The read operation performs better if called a single time.

However, the create and delete operations perform better when called several times as in the 100 files test runs. The delete operation achieves twice the throughput in the 100 file test runs than in the single file test runs. This is probably the result of internal caching in the MDS of Lustre. The MDS, for instance, caches several requests in memory before committing the requests to disk.

However, there are certain inconsistencies in the observed data that should be noted. For instance, in the 1GBit, 1 file, read command test run, the measured performance of the test setup with the client interceptor alone is 21.719 operations per second. However, the measured performance of the test setup with client and MDS interceptor is 22.270 operations per second. This is not reasonable and should not happen. The source of this error in the measurements might be changing occupation of the nodes due to other running processes in the background or different workload on the network during the individual test runs. Additionally, as demonstrated in the data, the advantage of the faster Gigabit network is much larger in the default Lustre setup. This result also suggests some problems with the correct adaptation of the interceptors to the file system.

In summary, the observed test values show some small inconsistencies, but nevertheless appear to be plausible. The major result of the test runs is the large performance overhead which the redesign places on the file system. This overhead makes the proposed redesign solution infeasible without some mitigation of the performance overhead. As noted above, the source of the significant increase in latency times is most likely in the file system code itself. To ascertain the reason for the performance impact and propose a workable solution, Lustre code needs to be examined and understood more completely. This is, unfortunately, not possible given the limited time allocated for this project.