MDS SMP Node Affinity Solution Architecture

Introduction

Current versions of Lustre rely on a single active metadata server. Metadata throughput may be a bottleneck for large sites with many thousands of nodes. System architects typically resolve metadata throughput performance issues by deploying the MDS on faster hardware. However, this approach is not a panacea. Today, faster hardware means more cores instead of faster cores, and Lustre has a number SMP scalability issues while running on servers with many CPUs. Scaling across multiple CPU sockets is a real problem: experience has shown in some cases performance regresses when the number of CPU cores is high.

Over the last few years a number of ideas have been developed to enable Lustre to effectively exploit multiple CPUs. The purpose of this project is to take these ideas past the prototype stage and into production. These features will be benchmarked to quantify the scalability improvement of Lustre on a single metadata server. After review, the code will be committed to the Master branch of Lustre.

Project Apus

Apus is a faint constellation in the southern sky, and represents the "bird of paradise" or "exotic bird". The Apus is also a genus of birds, commonly known as the swift, and are among the fastest birds in the world. This represents the goal to make the metadata server perform as fast as possible.

Solution Requirements

Vertical scalability of Lustre MDS is being improved, layer by layer, in the software stack. With Project Pisces Parallel Directory Solution Architecture shared directory performance was addressed for the (ldiskfs) backend filesystem. Project Apus will focus on improving SMP performance of other layers in stack.

Improve small message rate of LNet

The message rate of LNet is fundamental to metadata throughput. After detailed study and surveying, LNet message rate is judged to be limited by the unoptimized behavior on SMP architectures. These issues will be address to increase LNet small message rate. Improvements to LNet will be verified by LNet selftest.

Improve RPC rate of ptlrpc service

Overall metadata throughput is also dependent on the ptlrpc service. Gains from improving LNet message rate may not be realized because each ptlrpc service only has a single request queue, which means only a single CPU might be adding/removing request from the queue at a single point in time, so if many messages came at the same time, the network receiving threads would busy waiting to insert, and will insert one by one and at the same time servicing threads can fetch one by one too creating quite a bit of contention at the same time (i.e. cpus other than the one that

lucked out to get access to the queue would be sitting in a busy loop and prevent any other useful code to be running on those other cpus as well).

Project Apus will redesign the threading mode of the ptlrpc service and implement multiple request queues, SMP node affinity thread-pool for ptlrpc service will be introduced. RPC rate of ptlrpc service will be improved by this work. Improvements to the ptlrpc service will be verified by the Lustre echo_client.

Reduce overall threads count for Lustre

Without optimization, Lustre server on a multi-CPU system tends to spawn a large number of threads, for example, LND threads, ptlrpcd threads, workitem threads, and hundreds of service threads. If there are enough clients and all clients are trying to communicate with the same sever at the same time, number of messages arrived on that server will always be more than number of threads on the server, which means most threads could be woken up, system will have a very long runq and large overhead of context switches. Although it's difficult to decrease number of service threads (some requests could block service threads, server could hang if without enough threads), but we will systematically review the threads that are created in all Lustre modules and remove redundant threads.

Other SMP improvements for Lustre

After Project Apus has shown an increase in the RPC rate, attention will move to SMP locking. Highly contended global locks will be identified and readily available improvements will applied. In addition, key processing paths will be audited and any opportunities for cleanup will be taken.

With the sum of this work, metadata operation performance will increase on multi-socket SMP MDS.

Use Cases

Project Apus will benefit any Lustre deployment with a significant number of clients that is currently experiencing identifying the metadata server as a performance bottleneck. For example: a big MPI job which requires tens of thousands threads to parallel create or stat files, which means MDS will receive tens or hundreds of thousands requests in a few seconds, and all MDS services threads would be woken to handle these requests. In this case, performance of MDS typically suffers as lock contentions and unnecessary migrations of data between CPUs eat up useful clock cycles. Project Apus will improve performance of MDS by enhancing threading mode and locking mechanism.

Solution Proposal

The key idea for vertical scalability of Lustre is to improve parallel request handling wherever possible. Once parallel codes are developed, an additional idea is to schedule request handling on the nodes that have the relevant caches local.

Compute Partition

A multi-CPU server is divided into several processing partitions, each partition contains:

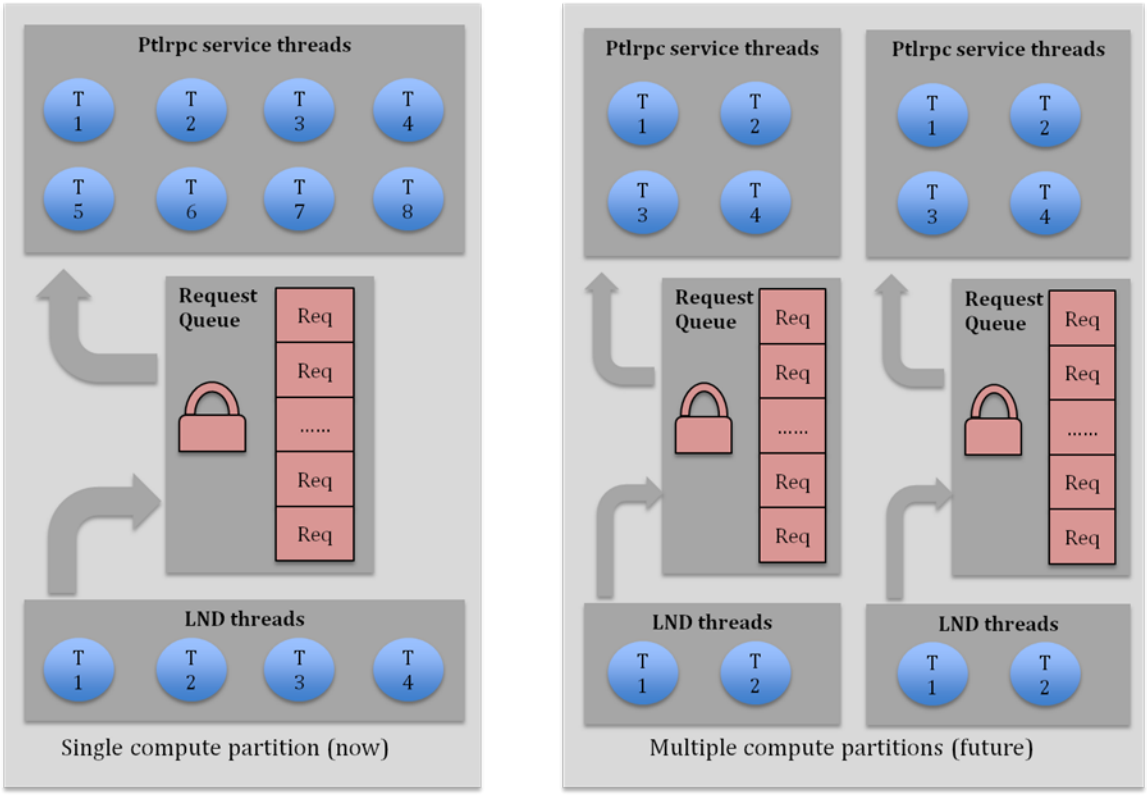
- A subset of CPUs (CPU partition)
- Memory pool
- Message queue
- Thread pool

Conceptually, these resources make up a simple virtual machine for Lustre requests, without unnecessary duplication of shared resources or connections that would otherwise be needed for dedicated virtual machines.

The MDS has a single namespace, however several virtual processing partitions (compute partitions) may exist. By design, compute partitions handle request with a minimum of interaction between each other. Minimizing interactions reduces lock contentions and cacheline conflicts.

As described above, each compute partition contains a non-overlapping subset of the CPU cores available (CPU partition). A CPU partition is conceptually similar to the 'cpuset' of Linux. However, in Project Apus, the CPU partition is designed to provide a portable abstract layer that can be conveniently used by kernel threads, unlike the 'cpuset' that is only accessible from userspace. To achieve this a new CPU partition library will be implemented in libcfs.

Graph-1 illustrates parallel request handling with multiple compute partitions. Although the figure on the right side only shows two compute partitions, more partitions are possible and it will be configurable depending on the hardware resources.



Graph-1: Compute partition of Lustre

Lustre threading mode

Currently, Lustre does not set CPU affinity for MDS service threads. This means threads could migrate between CPUs in a sub-optimum way. In addition, data context may be transferred between CPUs which further degrades performance.

Project Apus will address these concerns by providing services with a thread-pool for each compute partition. This differs from the existing implementation of a single, large thread-pool over all CPUs. Threads in a compute partition will have affinity to a CPU of that compute partition. This design is expected to reduce overhead incurred by thread and data migration between CPUs.

NUMA allocators

Threads in a compute partition could share some local data structure and buffer pool, on UMA system we can just

use regular memory allocators, but on NUMA system, we expect those threads can mostly use “local” memory, which should be managed by NUMA allocators. So we also need to add NUMA interfaces to libcfs.

LNet & LND changes

Besides general SMP improvements (locking improvements for LNet & LND, cacheline optimization, threading mode changes in LNDs), additional features will be added including:

- Parallelize EQ (LNet Event Queue) callback
  Currently, LNet only allows one thread to enter EQ callback at any instant.
- Localize buffer posted by LNetMDAttach
  Currently, All request buffers (wildcard receiving buffer) posted by LNetMDAttach are shared by all LND threads which is not good for performance because threads have to drag data between CPUs, we will change it in this project so buffers are most used by LND threads in that compute partition.

> **LNet API**
>
> There is no change for LNet API except a new flag for posting local buffer. This will be covered detail in the High Level Design document.

General SMP improvements of Lustre and LNet

SMP improvements have been made to Lustre with both 2.0 and 2.1 releases. However, many areas could be improved to continue to improve metadata performance including:

- Finer-grained locks.
- Hash function improvements.
- Cleanup logic overhead on hot processing path.
- Cacheline optimization.

Unit/Integration Test Plan

Verify LNet RPC rate improvement by LNet selftest

LNet selftest is a tool that runs on the LNet stack. It contains a lightweight RPC implementation based on LNet, a testing framework, and a userspace utility. LNet selftest can be used to benchmark both LNet bandwidth and RPC rate. 'ping' test of LNet selftest will be used to verify the improvement in LNet message rate.

Verify metadata performance improvement by mdtest or mdsrate

mdtest or mdsrate will be used to verify metadata performance improvement over whole Lustre metadata server software stack. The test case is similar to that of “Parallel directory operations” with the difference that metadata performance will be benchmarked for both “shared directory” and “directory per thread”.

Verify no functionality regression

This will be verified by our daily automated testing.

Acceptance Criteria

Apus will be accepted as properly functioning if:

- Improving metadata performance (creation/removal/stat) with these conditions:
    - Reasonable number of clients (16+ clients)
    - Fat MDS (8+ cores)
    - Narrow stripecount file only (0 – 4 stripecount)
- No performance regression for single client metadata performance
- No functionality regression

Glossary

SMP Symmetric multiprocessing, a computer hardware architecture where two or more identical processors are connected to a single shared main memory and are controlled by a single OS instance (Wikipedia.)