Administration of QoS for Lustre

Intel HPC Developer Conference 2017

L. Zeng, A. Brinkmann, T. Süß – JGU
L. Xi, Q. Yingjin, S. Ihara – DDN
Johannes Gutenberg University Mainz

- Founded in 1477 and reopened after a 150-year break in 1946 by the French forces
- 35,000 students from about 130 nations
- 4,150 academics, including 540 professors, teach and conduct research in JGU's more than 150 departments, institutes, and clinics
- Extraordinary research achievements in the fields of particle and hadron physics, materials sciences, and translational medicine
### Mogon II

Mogon II - MEGWARE MiriQuid, Xeon E5-2630v4 10C 2.2GHz, Intel Omni-Path

<table>
<thead>
<tr>
<th>Site:</th>
<th>Universitaet Mainz</th>
</tr>
</thead>
<tbody>
<tr>
<td>System URL:</td>
<td><a href="https://hpc.uni-mainz.de/high-performance-computing/mogonbild">https://hpc.uni-mainz.de/high-performance-computing/mogonbild</a></td>
</tr>
<tr>
<td>Manufacturer:</td>
<td>MEGWARE</td>
</tr>
<tr>
<td>Cores:</td>
<td>16,500</td>
</tr>
<tr>
<td>Linpack Performance (Rmax)</td>
<td>557.572 TFlop/s</td>
</tr>
<tr>
<td>Theoretical Peak (Rpeak)</td>
<td>580.8 TFlop/s</td>
</tr>
<tr>
<td>Nmax</td>
<td>2,534,400</td>
</tr>
<tr>
<td>Nhalf</td>
<td>220,000</td>
</tr>
<tr>
<td>Power:</td>
<td>242.43 kW [Submitted]</td>
</tr>
<tr>
<td>Memory:</td>
<td>81,920 GB</td>
</tr>
<tr>
<td>Processor:</td>
<td>Xeon E5-2630v4 10C 2.2GHz</td>
</tr>
<tr>
<td>Interconnect:</td>
<td>Intel Omni-Path</td>
</tr>
<tr>
<td>Operating System:</td>
<td>CentOS</td>
</tr>
</tbody>
</table>

### RANKING

<table>
<thead>
<tr>
<th>List</th>
<th>Rank</th>
<th>System</th>
<th>Vendor</th>
<th>Total Cores</th>
<th>Rmax (TFlops)</th>
<th>Rpeak (TFlops)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/2016</td>
<td>265</td>
<td>MEGWARE MiriQuid, Xeon E5-2630v4 10C 2.2GHz, Intel Omni-Path</td>
<td>MEGWARE</td>
<td>16,500</td>
<td>557.6</td>
<td>580.8</td>
<td>242.43</td>
</tr>
</tbody>
</table>
QoS for Lustre Project

The project is funded by Intel

- Intel Parallel Computing Center for Lustre
- Lustre QoS: Network Request Scheduler and Monitoring Revisited

Close cooperation with DDN

- Throughput for multiple flows
- Proportional Sharing Spare Bandwidth
Agenda

• Why do we need Quality of Service for HPC?
• Architectural Approaches
• IPCC for Lustre: Research and Development
• Administrators wish-list
• Open Problems
Why Quality of Service?

I/O resources are typically not part of the scheduling process

- Users might acquire bigger capacity share of the storage system, but do not receive more bandwidth
- Individual compute jobs are able to (accidentally) perform denial of service attacks by flooding the parallel file system with many small requests or metadata operations
- Concurrently running checkpoint operations overload parallel file system bandwidth and therefore prolong application runtimes
Why Quality of Service?

Different jobs require different treatment

• Throughput limited jobs require permanently data to process (e.g. ATLAS)

• Some applications create millions of files in (extremely) close succession and harm all other users

• Often checkpointing produces short write bursts in regular time distances
QoS Planning in Lustre

- QoS Planning for storage resources
  - Guarantee \( x \) GB/s read throughput
  - Guarantee \( y \) GB/s write throughput

- Architecture includes
  - Batch system
  - Client and/or server component in Lustre enforcing QoS
Architecture including QoS-Planner

Lustre Clients

- MDT
- MDS
- MGT
- MGS

- OSS
- OST
- OST
- OST
- OST
- OST
Architecture including QoS-Planner

Lustre Clients

- MDT
- MDS
- QoS Planner
- MGT
- MGS

- OSS
- Batch System
- Monitoring System
- OST
- OST
- OST
- OST
Architecture including QoS-Planner

Lustre Clients

- MDT
- MDS
- MGT
- MGS

QoS Planner

OSS

OST

Batch System

OSS

OST

Monitoring System
Initial Control Flow

1. Submit Job
   - /foo/compute
   - 12 cores
   - 12 GB memory
   - /foo/bar
   - 10 GB/s

2. Request

3. get_osts(/foo/bar)

4. [ost1, ost2]

5. Ack

6. Set NRS policies

MDS

QoS Planner

OSS
Token Bucket Filter (TBF)

Incoming Requests

Classifier

Dequeue request at class deadline

FIFO Queues
Token Bucket Filter (TBF)

- TBF is implemented inside Lustre’s Network Request Scheduler (NRS)
- 1 Token = 1 RPC ≈ 1 Mbyte (for 1 Mbyte chunks)
- Class-based TBF can classify by User ID, Job ID, ...
- Batch system / Administrator assigns token rates (per OSS)

- Enables fair bandwidth distribution
- Enables utilization of full bandwidth of OSSs

Results will be presented on SC 2017, Tuesday, November 14th, 11:30am, Location 405-406-407
“A Configurable Rule-Based Classful Token Bucket Filter Network Request Scheduler for the Lustre File System”
Clients with same rate receive same bandwidth
Results

Reservation is automatically adjusted when free bandwidth is available.
In the IPCC for Lustre researchers and HPC administrators work closely together

- Researchers develop solutions for problems
  - Cooperative development of the TBF; part of Lustre (DDN, JGU)
  - Development of QoS Planner for semi-automatic bandwidth control

- Administrators help to improve product
  - Deep knowledge of system requirements and challenges
  - Deployment and testing in large system
Admin´s wish list: Stability

- Must be reliable
- Must not harm other components
  - not system critical
  - system runs without QoS in standard configuration
- Not too complex
Solution: QoS Planner

- QoS Planner Frontend
- Server
- Coordinator
- Scheduler
- lctl and lfs
- OSSs
- MDS
Admin’s wish list: Flexibility

- Default configurations for user and/or job classes
- Changing scheduling policies
- Unexpected loads
  - RAID rebuilds
- Changing environment
  - rolling upgrades
  - monitoring systems
  - hardware changes/extensions
Solution: Replaceable Scheduler

- QoS Planner
- Frontend
- Server
- Coordinator
- Scheduler
- log
- lctl and lfs
- OSSs
- MDS
- Batch system
Admin’s wish list: Integration

- User accounting
- Monitoring
- Hardware environment
Solution: Replaceable Monitoring

QoS Planner
Frontend

Batch system

Server

Coordinator

Scheduler

log

Job States

Cluster State

Schedule State

Monitor

DB

lctl

lfs

OSSs

MDS

Slurm

QoS Planner

Coordinator

Scheduler

DB

Cluster State

Schedule State

Monitor

Solution: Replaceable Monitoring
Solution: Using LIME
Solution: Mountable Databases
Bandwidth is defined as a global and as a local resource.

- Slurm plug-in controls:
  - Globally available bandwidth - treated as license (one license/MB)
  - Local bandwidth - treated as generic resource

- Job gets rejected if one resource is not available

- Example:
  ```bash
  srun -N1 -gres=qoslustre:100M -L lustreqos:100 sleep 5
  ```
Putting it all together ...

We have integrated our QoS-Planner on our productive system Mogon II

- QoS server runs on two OSSs responsible for scratch file system
  - nrs_policies="tbf jobid"
  - jobid_var=procname_uid
- OSS use Lustre’s TBF version 2.8
- QoS client is installed on compute nodes
  - jobid_var=procname_uid
A client application for reserving bandwidth has been developed for Slurm

```
# qosp reserve -throughput 100 -duration 100 \ 
-filenames /path/to/folder
```

- Command reserves a **throughput** of 100 RPCs for 100 seconds
- OSTs are identified via **filenames** respectively paths
- Available shares can be identified via **id**
Slurm-plugin uses `qosp` command for reserving bandwidth. Throughput is taken from global and local resources.

Further integrations are possible:

- Coupling of users or groups with QoS manager:
  - Groups that gave additional money for storage get more shares
  - Malicious users/groups can be throttled down
- Credit bandwidth of reservations that terminate earlier
Spontaneous Reservation

Many programs require high I/O bandwidth only for a short time period
• Loading input data during initialization
• Checkpointing
• Storing final results

We provide a C++ API for spontaneous I/O accesses
• Reserve bandwidth for a certain time span
• Test if reservation is available
• Remove reservation after I/O is done
Most important API functions:

// none-blocking reservation
string addReservationAsync(int tp, int sec, string fs);

// blocking reservation
string addReservationSync(int tp, int sec, string fs);

// delete a specific reservation
bool removeReservation(string id);

// test the status of a reservation
// (UNDEFINED, SCHEDULED, ACTIVE)
// required for asynchronous reservation
int testReservation(string id);
Spontaneous Reservation

QoS scheduler can use e.g. backfilling, thus a reservation start time may change

Asynchronous functions supports this behavior

```
// none-blocking reservation
string addReservationAsync(int tp, int sec, string fs);
// test the status of a reservation
int testReservation(string id);
```
Open Problems

Authentication is required
• Self-made authentication
• Using ssh with public keys

First release
• More hardening and testing required
• Beta available
Thank you for your attention

Questions?