Analyzing Characteristics of Parallel Jobs on the Mistral Supercomputer

Eugen Betke, Julian Kunkel

Research Group
German Climate Computing Center

Workshop on Storage Challenges in the UK
March 6, 2019

Table of contents

Table Of Content

Goals

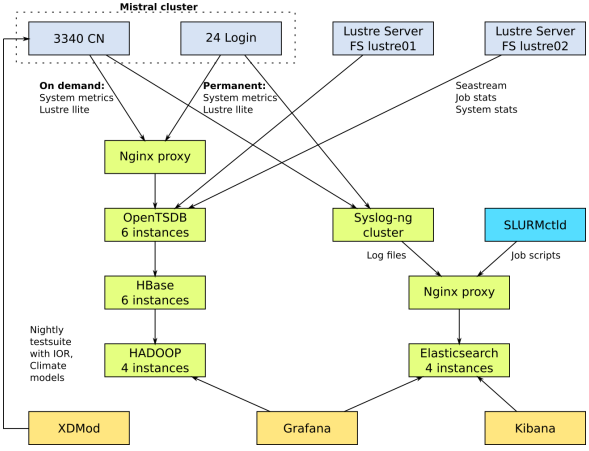
Motivation

- Understanding the workload of the Mistral Supercomputer.

Goals

- Monitoring system development
  - A flexible and extensible monitoring system
  - A portable solution for the next HPC generation
- Establishing analysis workflows
  - Identification of problematic applications and key workloads
  - Understanding of typical I/O patterns
- Tooling
  - Automatic identification of inefficient applications (long term goal)

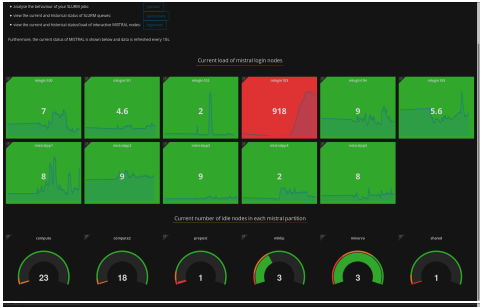
DKRZ Supercomputer and Monitoring



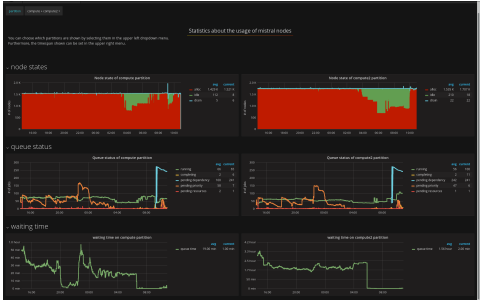
- The Mistral Supercomputer
  - 3,340 client nodes
  - 24 login nodes
  - 2 Lustre file systems
  - Slurm workload manager
- Monitoring System is built of
  - open source components
  - a self-developed data collector
- **Provides statistics about**
  - **login nodes**
  - **user jobs**
  - **workload manager queue**

User Interface: Grafana with Templates

Overview



Slurm



- These views are always available
- Job monitoring is optional
  - Can be enabled by a Slurm parameter

Login nodes

Envisioned User Workflow

- Visualization of system behavior
  - Coarse grained monitoring is always active
  - On demand refinement of the job monitoring
  - Options to change the monitoring behavior
- Visualization of specific jobs
  - Must be explicitly enabled by a Slurm parameter
  - After job finishes the monitoring data is available in Grafana
    - A view in Grafana provides a list of monitored jobs
    - The data is available for 24 hours

Refining Monitoring Settings

- Slurm monitoring options and capture interval
  - **cpu** : cpu usage (usr, system, idle, iowait)
    - gathered by sockets to e.g. identify non optimal CPU binding, cpu frequency per core/HT to identify throttling
  - **meminfo** : memory usage (available, cached, active)
    - allowing to immediately recognize leaks in application
  - **lustre** : read/write bytes per second
  - **power** : energy consumption

Example

```
#!/bin/zsh
#SBATCH -N 4
#SBATCH --monitoring=meminfo=10,lustre=5,cpu=5,power=5

srun -N 4 --ntasks-per-node=4 ior -s 10000 -b 2097152 -t 2097152 -a POSIX
```

User Interface: Job Statistics

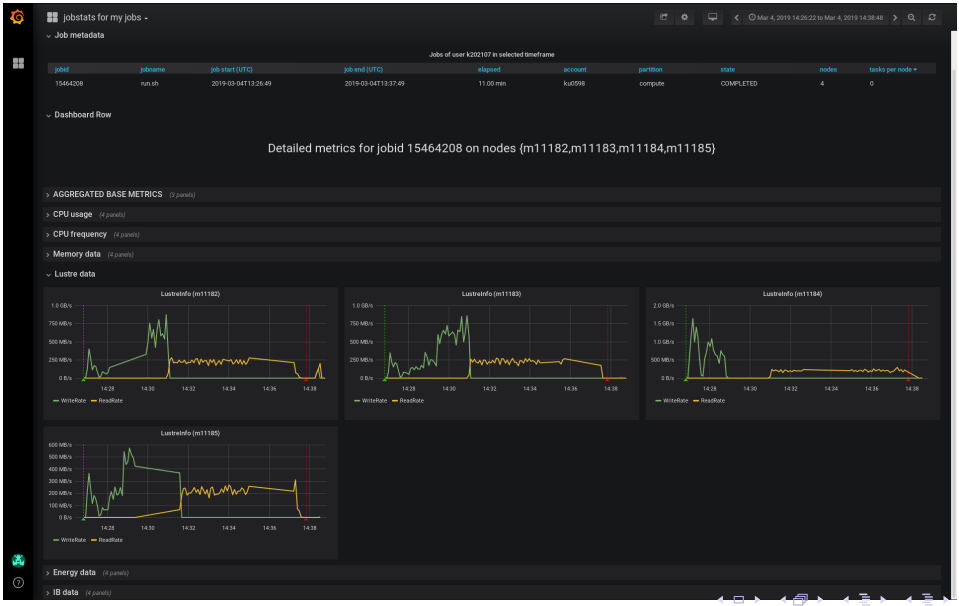


Table Of Content

Captured I/O Metrics

- I/O metrics are captured and archived by default for each job
- Some metadata metrics are cumulated
- Relevant Lustre metrics are captured (focus on key aspects)

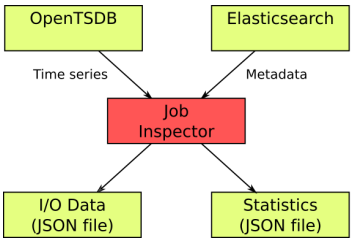
Source file: /proc/fs/lustre/llite/lustre*-*/stats

```
md_read = getattr + getxattr + readdir + statfs + listxattr + open + close
md_mod = setattr + setxattr + mkdir + link + rename + symlink + rmdir
md_other = truncate + mmap + ioctl + fsync + mknod
md_file_create = create
md_file_delete = unlink
```

Source file: /proc/fs/lustre/llite/lustre*-*/read_ahead_stats

```
osc_read_bytes, osc_read_calls
osc_write_bytes, osc_write_calls
read_bytes, read_calls
write_bytes, read_calls
seek
```

Analysis Workflow



- A daemon process iterates over all jobs
- Each job is fetched and analysed
  - Elasticsearch provides meta data
  - OpenTSDB provides I/O time series
- Output
  - I/O data is stored in separate JSON files
  - Statistics (Object of investigation)
    - Currently done in R
- The tool is in an early development stage

Aggregated Data in JSON format: a Sample


```
{
 "metadata": {
   "_source": {
     "time_limit": 5400,
     "@end": "2018-12-04T11:32:23",
     "cpu_hours": 0.057778,
     "cpus_per_task": 1,
     "total_cpus": 8,
     "@eligible": "2018-12-04T11:31:23",
     "elapsed": 26,
     "jobid": 14407499,
     "state": "COMPLETED",
     "jobname": "mkm0003_post",
     "ntasks_per_node": 8,
     "@start": "2018-12-04T11:31:57",
     "ntasks": 8,
     "groupname": "mpioes",
     "nodes": " m11515 ",
     "job_name": "mkm0003_post",
     "user_id": 23738,
     "group_id": 2100,
     "exit_code": "0:0",
     "total_nodes": 1,
     "account": "ba0989",
     "username": "m300467"
   }}}
```

```
"ts": {
 "read_bytes": [
   {
     "metric": "host.lustre.stats.read.bytes",
     "dps": {
       "1515756295": 5104980744214,
       "1515756305": 5104980753366,
       "1515756310": 5104980867566,
       "1515756290": 5104980741946,
       "1515756300": 5104980753366
     },
     "aggregateTags": [],
     "tags": {
       "name": "lustre01",
       "system": "mistral",
       "host": "m10753"
     }
   }
 ]
}
```

Table Of Content

General Information about the Test Dataset

- Data from 5 day (from 2018-12-07 to 2018-12-13)
- 70846 jobs statistics downloaded in JSON format (360GB)
- **33193 (47%) jobs are evaluated**, that
  - contain non-empty time series and
  - and have exit status `COMPLETED`
- 203 users $\rightarrow$ 183 jobs per user in average

Exit status statistics

1026	CANCELLED
63636	COMPLETED
5753	FAILED
3	NODE_FAIL
426	TIMEOUT

Derived Metrics

- New metrics
  - that **provide more information**
  - e.g. "bytes/call" for read and write
- Other job characteristic metrics
  - to **identify I/O intensive jobs**
  - e.g. data read and written by a node
- Independent metrics
  - that can be used to **compare jobs**
  - e.g. average call rate done by a process

Examples: write metrics

metric	Description
'write_bytes'	Data written (job)
'write_bytes_nn'	Data written (node)
'write_bytes_ppn'	Data written (process)
'write_bytes_rate'	I/O performance (job)
'write_bytes_nn_rate'	I/O performance (node)
'write_bytes_ppn_rate'	I/O performance (process)

Example: Statistics for an I/O intense Job

General information

Nodes	4
Processes	7
Elapsed time	4.8h

Metadata access

Metadata read ops	27 Mops
Metadata read frequency	**1537 ops/s**

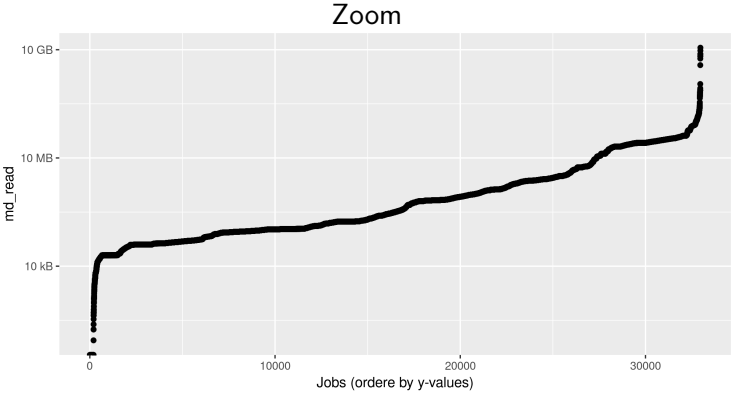
Read

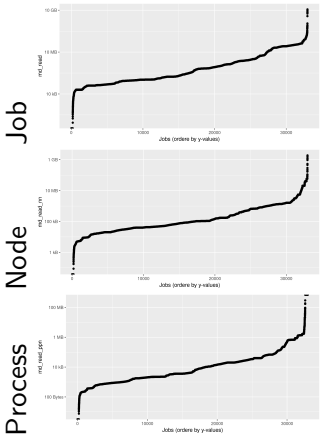
(Avg.) Bytes/op	**4 MB**
Total data	78 TB
(Avg.) Performance (job)	4.5 GB/s
(Avg.) Performance (node)	**1.1 GB/s**
(Avg.) Performance (process)	0.2 GB/s
Operations	19.3 Mops
(Avg.) Operation frequency	1108 ops/s

Write

(Avg.) Bytes/op	**3.3 MB**
Total data	71 TB
(Avg.) Performance (job)	4.1 GB/s
(Avg.) Performance (node)	**1.0 GB/s**
(Avg.) Performance (process)	0.1 GB/s
Operations	21.2 Mops
(Avg.) Operation frequency	1218 ops/s

Looking at Metadata

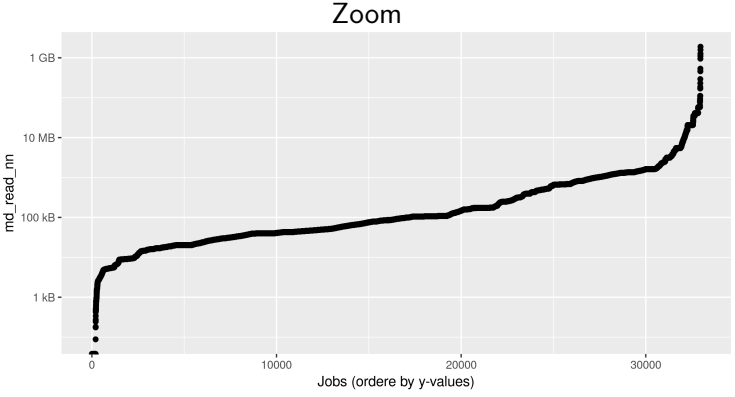


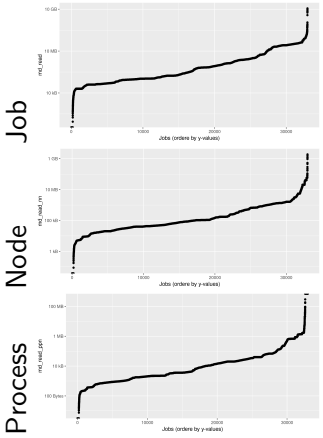


Source file: `/proc/fs/lustre/llite/lustre*-*/stats`

```
md_read = getattr + getxattr + readdir + statfs + listxattr + open + close
```

Looking at Metadata

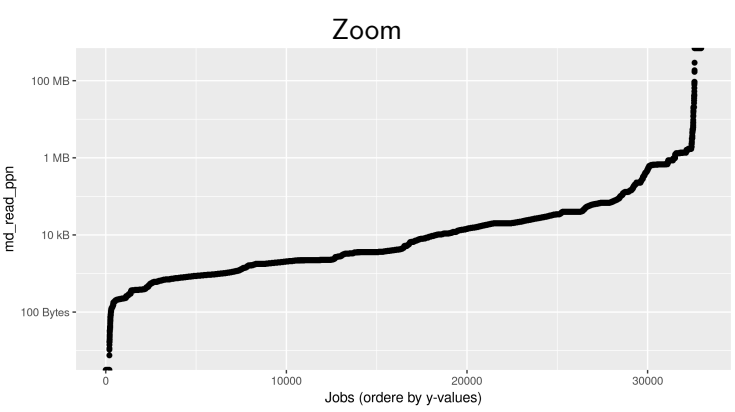


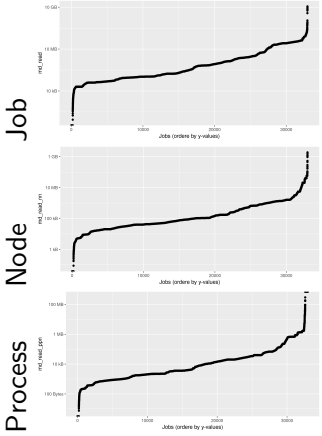


Source file: `/proc/fs/lustre/llite/lustre*-*/stats`

```
md_read = getattr + getxattr + readdir + statfs + listxattr + open + close
```

Looking at Metadata





Source file: `/proc/fs/lustre/llite/lustre*-*/stats`

```
md_read = getattr + getxattr + readdir + statfs + listxattr + open + close
```

Example of Ranking Across Jobs and Users

Who are the top 5 jobs and users, doing the most meta data reads?

Jobs

jobid	md_read Millions	job_name	username
144U	11,371	fw	u2
144V	9,475	fw	u2
144X	7,583	fw	u2
144Y	7,579	fw	u2
144Z	6,640	fw	u2

Users

username	md_read / job Millions
u2	5094
b3	234
m3	187
a2	118
m2	106

Table Of Content

Summary

- DKRZ monitoring system
  - **Open source** components + self-developed collector
  - **Portable** to the next HPC and other machines
- Analysis approach
  - **Ranking** of metrics of interest
- First analysis allows us to identify
  - **Identify large workloads**
  - **Find users** who create large workloads
  - **Compare jobs**