Machine Learning of I/O Behavior

Eugen Betke, Julian Kunkel

Research Group
German Climate Computing Center

Open Source AI Workshop
5th April 2019

Table of contents

1. DKRZ Monitoring
2. Statistical Analysis
3. Job Footprinting
4. First experiments
5. Summary

Table Of Content

1. DKRZ Monitoring
2. Statistical Analysis
3. Job Footprinting
4. First experiments
5. Summary

Goals

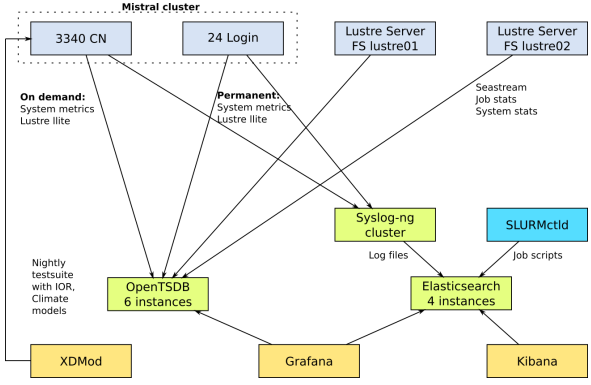
Motivation

- 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**

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)

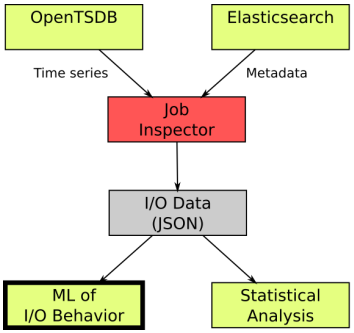
Source file: /proc/fs/lustre/llite/lustre*-*/stats

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

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
  - Job statistics
  - Sequence of I/O behaviour
- 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": 14407,
     "state": "COMPLETED",
     "jobname": "mkmpost",
     "ntasks_per_node": 8,
     "@start": "2018-12-04T11:31:57",
     "ntasks": 8,
     "groupname": "mpis",
     "nodes": " m11515 ",
     "job_name": "mkmpost",
     "user_id": 237,
     "group_id": 210,
     "exit_code": "0:0",
     "total_nodes": 1,
     "account": "ba09",
     "username": "m3"
   }}}
```

```
"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

1 DKRZ Monitoring

2 Statistical Analysis

3 Job Footprinting

4 First experiments

5 Summary

Statistical Analysis: 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)

Statistical Analysis: Overview Dashboard

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

Statistical approach

Cans

- Provides understandable representation of job data
- Shows many useful information, that allows
  - Identify high work loads and responsible users
  - Compare jobs

Cant's

- Provides average I/O values only
  - can't always identify bad I/O performance
  - e.g. I/O phases can be short, but fast
- Doesn't consider execution phases
  - e.g. creating checkpoints, computing, reading/writing data, ...

Table Of Content

1 DKRZ Monitoring

2 Statistical Analysis

3 Job Footprinting

4 First experiments

5 Summary

Goal: Mathematical Representation of I/O Data

- Mapping of captured job data to a fixed length vector
- Each element represents weighted I/O behaviour

Goal

$$\text{footprint}(\text{jobid}) = \vec{v}_{\text{jobid}} \tag{1}$$

with $\vec{v}$ is a fixed length numeric vector

Goal: Mathematical Representation of I/O Data

- Mapping of captured job data to a fixed length vector
- Each element represents weighted I/O behaviour

Example

$$\text{footprint}(14400233) = \begin{pmatrix} X1:3 \\ X2:1 \\ X3:3 \\ X4:1 \end{pmatrix} \quad (1)$$

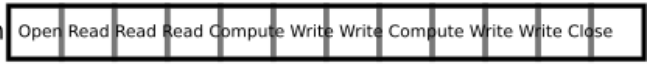
I/O Behavior

X1:	Metadata intensive
X2:	Using I/O node
X3:	Highly parallel I/O
X4:	No I/O

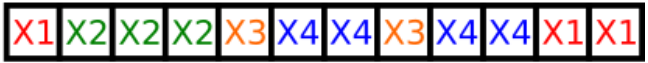
Basic Approach

1. Capturing

2. Segmentation



3. Classification



4. Labeling

X1 : 4 (Meta Data Access)

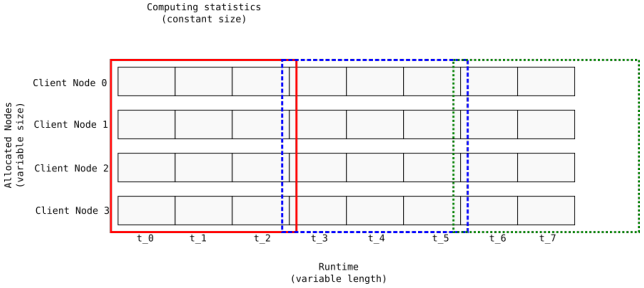
X2 : 2 (Read Intesity)

X3 : 2 (No I/O Phase)

X4 : 4 (Write Intesity)

Segmentation

- Problem
  - Number of nodes is variable
  - Segment size to large / too many segments
- Solution
  1. Split data in 2D segments
  2. Convert to $n \times n$ matrix
     - for each segment and
     - for each metric

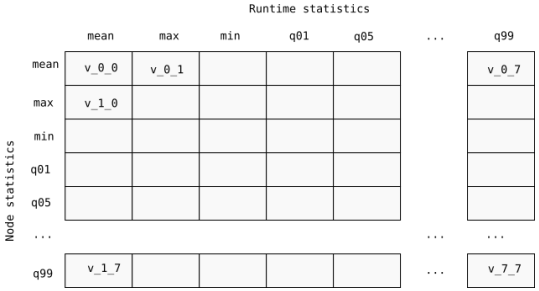


Conversion of Variable Length Vectors to Fixed Length Statistics

$$\text{stats}\left(\vec{v}\right) = \begin{pmatrix} \text{min} \\ \text{max} \\ \text{mean} \\ \text{q01} \\ \text{q10} \\ \text{q05} \\ \text{q25} \\ \text{q50} \\ \text{q75} \\ \text{q90} \\ \text{q95} \\ \text{q99} \end{pmatrix} \tag{2}$$

Computing statistics

- Statistics are organized as a 2D-matrix
- stats $(\vec{v})$ is applied to both axis
  - x-axis combines runtime
  - y-axis combines nodes
- The computation is done
  - for each segment
  - for each of 13 metrics



Resulting segment size after conversion

12 stats on x-axis * 12 stats on y-axis * 13 metrics = 1872 floating point values

Table Of Content

1 DKRZ Monitoring

2 Statistical Analysis

3 Job Footprinting

4 First experiments

5 Summary

General Information about the Test Dataset

- Data from 5 days
  - from 2018-12-07 to 2018-12-13
- 70846 jobs statistics downloaded in JSON format
  - uncompressed size is 360GB
- **33193 (47%) jobs are evaluated**, that
  - contain non-empty time series and
  - and have exit status `COMPLETED`

Exit status statistics

JOBS	EXIT STATUS
1,026	CANCELLED
63,636	**COMPLETED**
5,753	FAILED
3	NODE_FAIL
426	TIMEOUT

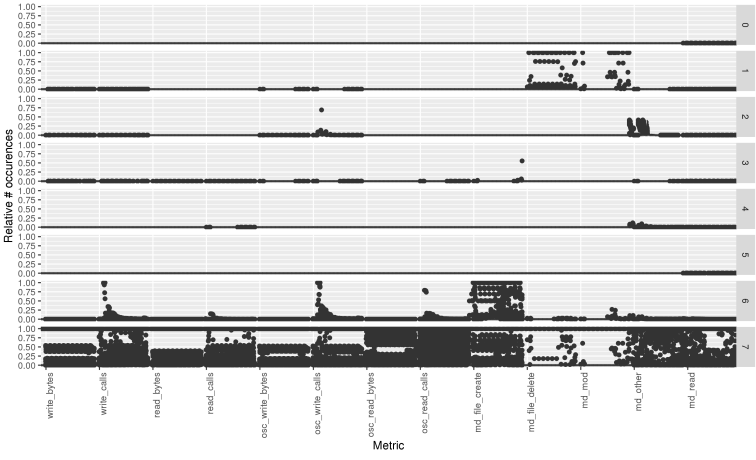
Slurm statistics

JOBS	SLURM PARTITION
37,989	**compute,compute2**
241	gpu
828	miklip
34	minerva
31,752	shared,prepost

Segmentation Parameters

- Data from 33,193 jobs is converted to 3,231,014 segments
  - Each segment is 1 minute long
  - Shorter segments are dropped
- Algorithm: kMeans (batch mode)
  - Input: segments (segment size is 1,872)
  - Output: 8 clusters

Segmentation Categories



Cat	Percent	Jobs
0	25.55	825,650
1	0.10	3,163
2	0.74	24,060
3	0.01	253
4	0.02	574
5	5.97	193,000
6	53.14	1,717,006
7	14.46	467,315

Cat	Description
0	No I/O
1	MD delete/modify
2	MD other intensive
3	Light read/write
4	Light MD other
5	No I/O
6	File create/inefficient write
7	Intensive I/O

Footprints (Example)

- Now a job can be represented as a sequence of I/O behavior.

Example

- Sequence of I/O behavior

```
[6 7 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 6 6 6 6 6 6 6 7
6 6 6 6 7 7 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 7 6 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 1 6 6 6 7 6 6 6 6 7 7 7 0 6 0 5 6 6 6 6 6 6
6 6 6 6 6 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 0 6 6 0 0 6 0 6 0 7 6 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 6 6 6 6 6 7 6 6 6 7 6 6 6 6 6 6
6 7 6 6 6 6 6 6 6 6 6 7 6 6 6 7 6 6 6 6 7 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 6 6 6 6 7 6 6 7 6 6 7 6 6 6 6 5 0 0 7 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 7 0 6 0 7 0 6 0 0 0 0 6 6 6 5 0 0 5 7 5 5 7 0 0 0 0 0 5 5 6 0 0 0 7 0
7 0 0 0 5 0 0 0 7 0 0 0 0 0 0 0 0 0 5 0 0 5 0 5 0 5 0 0 5 0 0 0 0 0 0 0 0
0 7 0 0 7 5 0 7 5 0 0 5 0 0 5 0 5 0 0 5 0 0 0 0 5 0 0 5 0 0 0 5 0]
```

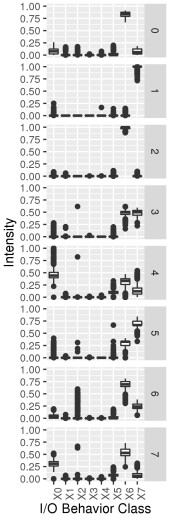
- Footprint

```
footprint(14461299) = [78, 1, 0, 0, 0, 24, 299, 38]
```

Footprint Clustering Parameters

- Footprints for each of 33193 jobs are created
- Footprints are normalized, to make them independent to job length
  - e.g. norm([78, 1, 0, 0, 0, 24, 299, 38]) = norm([156, 2, 0, 0, 0, 48, 598, 76])
- Algorithm: kMeans
  - Input: Footprints
  - Output: 8 clusters

Footprint Categories and Distribution



Footprint statistics

Cat	Percent	Jobs
0	13.32	4,192
1	6.06	1,906
2	39.34	12,384
3	7.27	2,290
4	6.25	1,968
5	9.75	3,069
6	8.91	2,805
7	9.10	2,864

I/O behavior

Cat	Description
X0	No I/O
X1	MD delete/modify
X2	MD other intensive
X3	Light read/write
X4	Light MD other
X5	No I/O
X6	File create/inefficient write
X7	Intensive I/O

Table Of Content

1 DKRZ Monitoring

2 Statistical Analysis

3 Job Footprinting

4 First experiments

5 Summary

Summary

- DKRZ monitoring system
  - **Open source** components + self-developed collector
  - **Portable** to the next HPC and other machines
- Statistical approach is a good way to
  - **Identify large workloads**
  - **Find users** who create large workloads
  - **Compare jobs**
- Job-Footprinting
  - **Categorization** of jobs