Statistical File Characterization & Status Update: Monitoring at DKRZ

Julian M. Kunkel

kunkel@dkrz.de

2016-11-17

Outline

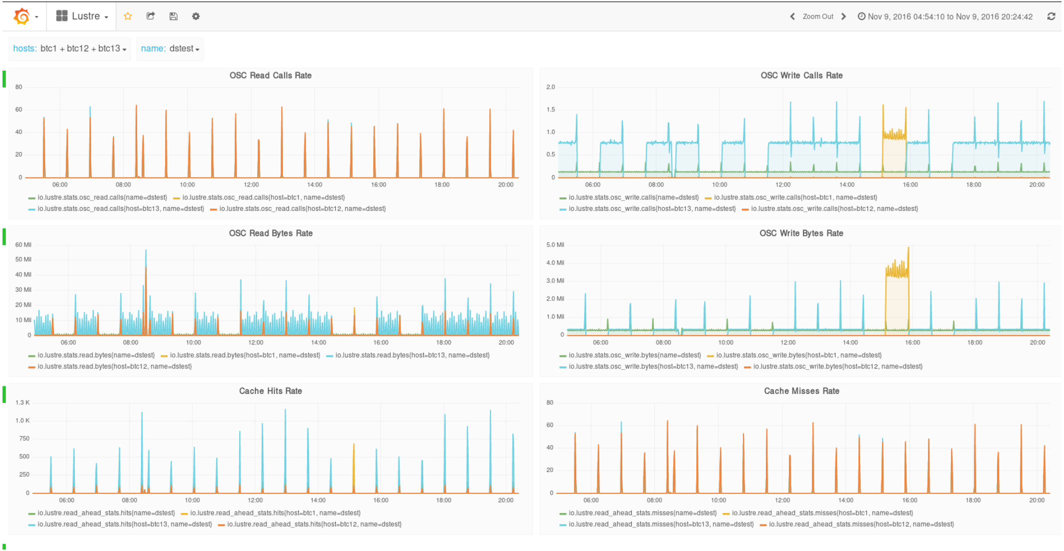
1 Status Update: Monitoring at DKRZ

2 Statistical Sampling

Status Update: Monitoring at DKRZ

- Monitoring backend OpenTSDB, Grafana
- Lustre OSS data: per job statistics using SLURM JOBIDs
- Additional information fed into the system
  - Site: energy consumption, temperature
  - Slurm: job information
  - Nodes: data from /proc (ongoing effort)
- We aim to support application-specific data
  - User-defined metrics, i.e., users can provide additional metrics
  - Based on automatic instrumentation, e.g., using SIOX to aggregate details
    - Why not have a per-file statistics in Grafana using I/O monitoring tools?
    - Why not show performance outliers according to an I/O model?
    - Number of online optimizations performed in SIOX

Grafana System View



Lustre

Zoom Out

Nov 9, 2016 04:54:10 to Nov 9, 2016 20:24:42

hosts: btc1 + btc12 + btc13

name: dstest

OSC Read Calls Rate
80
60
40
20
0
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
io.lustre.stats.osc_read.calls{name=dstest}
io.lustre.stats.osc_read.calls{host=btc1, name=dstest}
io.lustre.stats.osc_read.calls{host=btc13, name=dstest}
io.lustre.stats.osc_read.calls{host=btc12, name=dstest}
OSC Write Calls Rate
2.0
1.5
1.0
0.5
0
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
io.lustre.stats.osc_write.calls{name=dstest}
io.lustre.stats.osc_write.calls{host=btc1, name=dstest}
io.lustre.stats.osc_write.calls{host=btc13, name=dstest}
io.lustre.stats.osc_write.calls{host=btc12, name=dstest}
OSC Read Bytes Rate
60 Mil
50 Mil
40 Mil
30 Mil
20 Mil
10 Mil
0
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
io.lustre.stats.read.bytes{name=dstest}
io.lustre.stats.read.bytes{host=btc1, name=dstest}
io.lustre.stats.read.bytes{host=btc13, name=dstest}
io.lustre.stats.read.bytes{host=btc12, name=dstest}
OSC Write Bytes Rate
5.0 Mil
4.0 Mil
3.0 Mil
2.0 Mil
1.0 Mil
0
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
io.lustre.stats.osc_write.bytes{name=dstest}
io.lustre.stats.osc_write.bytes{host=btc1, name=dstest}
io.lustre.stats.osc_write.bytes{host=btc13, name=dstest}
io.lustre.stats.osc_write.bytes{host=btc12, name=dstest}
Cache Hits Rate
1.3 K
1.0 K
750
500
250
0
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
io.lustre.read_ahead_stats.hits{name=dstest}
io.lustre.read_ahead_stats.hits{host=btc1, name=dstest}
io.lustre.read_ahead_stats.hits{host=btc13, name=dstest}
io.lustre.read_ahead_stats.hits{host=btc12, name=dstest}
Cache Misses Rate
80
60
40
20
0
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
io.lustre.read_ahead_stats.misses{name=dstest}
io.lustre.read_ahead_stats.misses{host=btc1, name=dstest}
io.lustre.read_ahead_stats.misses{host=btc13, name=dstest}
io.lustre.read_ahead_stats.misses{host=btc12, name=dstest}

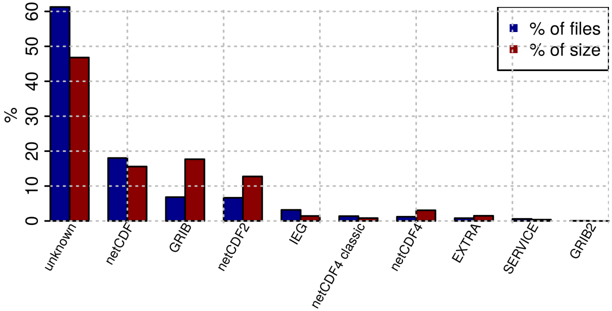
Motivation

- Understanding data characteristics is useful
  - Relation of file types to optimize relevant file formats
  - Conducting what-if analysis
    - Influence of compression, deduplication
    - Performance expectations for parallel I/O
- Analysing large quantities of data is time consuming and costly
  - Scanning petabytes of data in $>$ 100 millions of files
  - With 50 PB of data and 5 GiB/s read at least 115 node days are needed

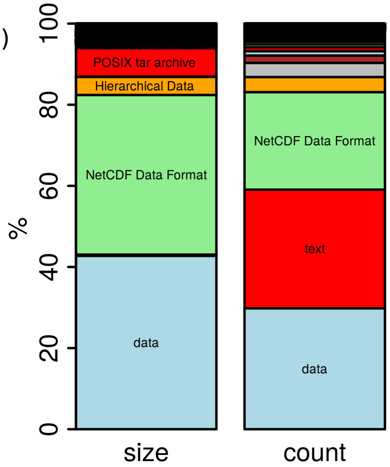
  $\Rightarrow$ Working on a representative data set reduces time and costs
- Conducting analysis on representative data is difficult
  - What data makes up a representative data set?
  - How can we infer knowledge for all data based on the subset?
    - Based on file numbers (i.e., a typical file is like X)
    - Based on capacity (i.e., 10% of storage capacity is like Y)

Computing Characteristics on Count/Size

- Example: scientific file formats (determined with CDO/file)
- The computation by file count and capacity differs
  - A heavy-tailed distribution of file sizes skews analysis



(a) CDO types



(b) File types

Tool to Perform Statistical Sampling

Methodology

- Investigation of statistical sampling to estimate file properties
  - Demonstrate the approach by applying statistical simulation
- Development of a tool (set of scripts) as startpoint to perform your studies
  Prototype: `https://github.com/JulianKunkel/statistical-file-scanner`
- It implements sampling to compute multiple metrics by capacity

Sampling Strategies

Sampling to Compute by File Count

1. Enumerate all files
2. Create a simple random sample
   - Select a random number of files to analyze without replacement
   - For proportional variables, the number of files can be computed with Cochran's formula
   - You can use simulation to estimate the error for contiguous variables

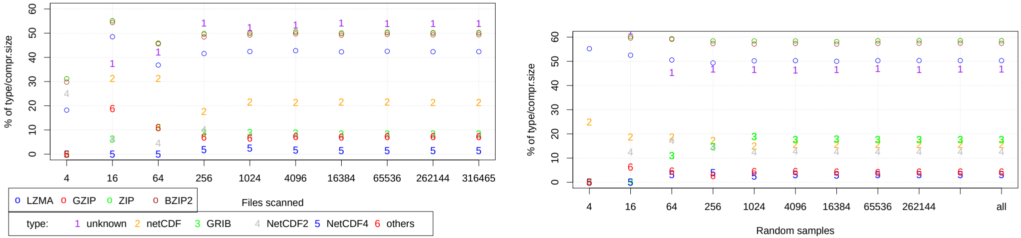
Sampling to Compute by File Size

1. Enumerate all files AND determine their file size
2. Pick a random sample based on the probability $\frac{filesize}{totalsize}$ with replacement
   - Large files are more likely to be chosen (even multiple times)
3. Create a list of unique file names and analyze them
4. Compute the arithmetic mean for the variables
   - If a file has been picked multiple times in Step 2., its value is used multiple times

Demonstration of the Strategies

- Apply the approach with an increasing number of samples
  - Compare true value with the estimated value

Running one simulation for increasing sample counts



(c) Compute mean by count

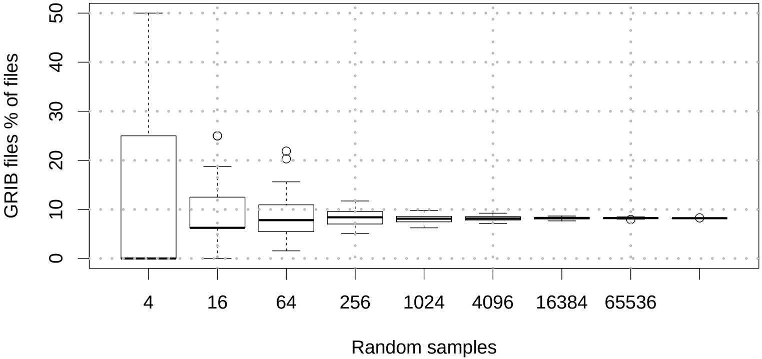
(d) Compute mean by size

Evaluating various metrics (proportions) for an increasing number of samples

- This suggests that the results converge quickly but how trustworthy is one run?

Investigating Robustness: Computing by File Count

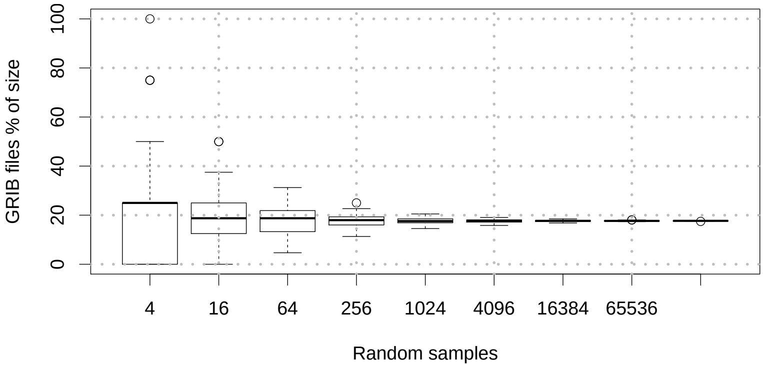
- Running the simulation 100 times to understand the variance of the estimate
- Clear convergence: thanks to Cochran's formula the total file count is irrelevant



Simulation of sampling by file count to compute compr.% by file count

Investigating Robustness: Computing by File Size

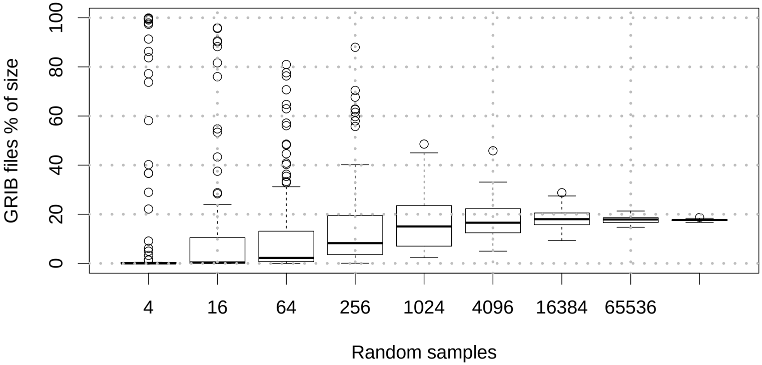
- Using the correct sampling by weighting probability with file size



Simulation of sampling to compute proportions of types by size

Investigating Robustness: Computing by File Size

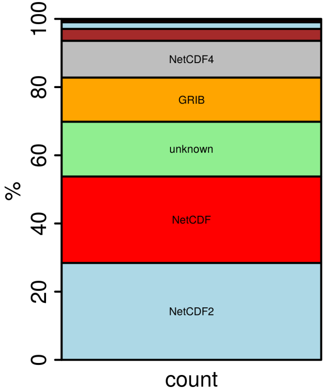
- Using the WRONG sampling by just picking a simple random sample
- Almost no convergence behavior; you may pick a file with 99% file size at the end



Simulation of sampling to compute proportions of types by size

Applying the Tool to DKRZ Complete Data

- Short study for lz4fast
- Two weeks runtime on one dual socket node
  - Scanned 11k files, 8 TByte
- Ratio: 0.68 ⇒ saving of 30% possible
- Speed in MiB/s:
  - 1059 (compression)
  - 1506 (decompression)
- Speed (memcpy): 900 (per core)
- Note that the way the compression threads are started influences the performance as the problem is memory bound



Summary

- Statistical sampling helps to quantify characteristics by file count and file size
- Samplying by file size requires to capture file list and determine file size
- Even with few samples a good accuracy can be achieved (e.g., 1% of file count/capacity)
- For further reading on the method:
  - Analyzing Data Properties Using Statistical Sampling Techniques–Illustrated on Scientific File Formats and Compression Features. Julian Kunkel. International Conference on High Performance Computing. 2016. Springer. pp.130-141
  - Analyzing Data Properties using Statistical Sampling–Illustrated on Scientific File Formats. Julian Kunkel. Supercomputing Frontiers and Innovations. 2016. Vol. 3(3). pp.34-39.

BACKUP

Statistical Sampling

- Can we determine the error when analyzing only a fraction of data?
  - We simulate sampling by drawing samples from the totally analyzed files
- Statistics offers methods to determine confidence interval and sample size
- We analyze random variables for quantities that are continuous or proportions
- Proportions: fraction of samples for which a property holds

Sample size and confidence intervals

- For proportions Cochran's sample size formula estimates sample size
  - (Similar number) works for extremely large population sizes
  - Error bound $\pm 5\%$ requires 400 samples (95% confidence)
  - Error bound $\pm 1\%$ requires 10,000 samples
- For continuous variables
  - Models require to know the distribution of the value
  - A-priori unknown, usually not Gaussian, difficult to apply → out-of-scope (here)
  - Nevertheless, we will demonstrate convergence

Sampling of the Test Data

- DKRZ usage: 320 million files in 12 PB, 270 project dirs
- Scan of user accessible data (scan is done by a regular user)
  - Accessible data: 58 million files, 160 project dirs
- Scanned files: 380 k files (0.12%) in 53.1 TiB (0.44%) capacity
  - Discrepancy since home directories contain very small files

Scanning Process

1. Run a find for each project directory, store it is a file
2. Select up to 10 k files from each project randomly (scan list)
3. Permutate the scan list
4. Partition the scan list into chunks (file lists)
5. Run multiple processes concurrently, each working on a file list
6. Terminate the processes after a couple of days

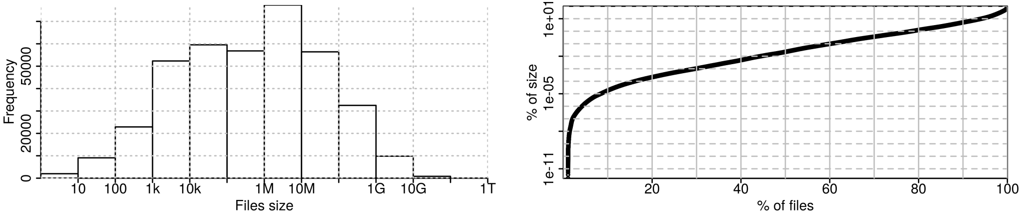
As we will see this approach is not optimal for analyzing by capacity

Processing of Files

- Processing is implemented as simple shell script
  - Starter script spawns multiple scripts each working on a file list
- The process reads the file list and current progress
  - Allows to restart processes and continue processing
- Process ignores non-existing files
  - Some files are deleted after the scan
- Copy the file
  - (Best alternative, copy data into memory)
  - You would not believe how many files changed on the fly
  - Running different tools on the data falsifies results
- Run tools to investigate properties:
  - `file` to identify file type (based on magic), suboptimal
  - CDO to identify scientific file format (high accuracy)
  - Compression tools: LZMA, GZIP, BZIP2, ZIP

Distribution of File Sizes

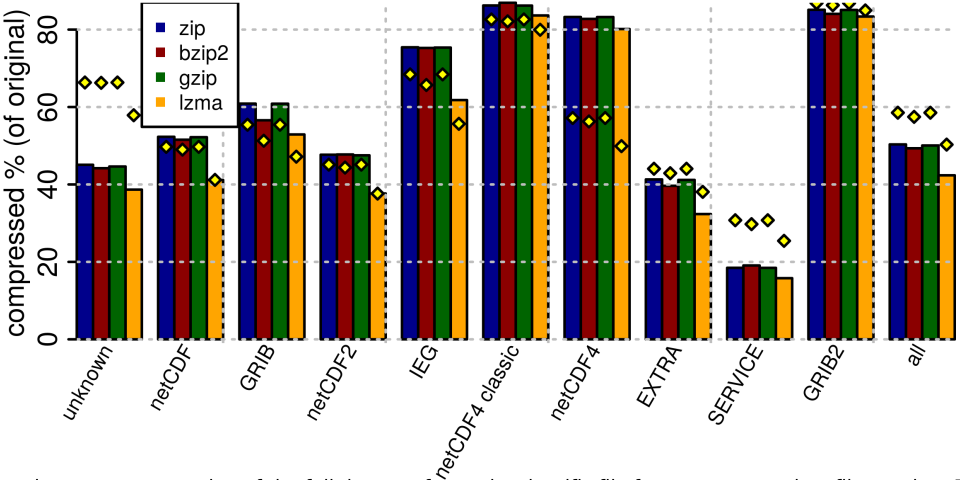
- For now, we analyze all scanned files!
- File size follows a heavy tailed distribution
- 90% of files consume roughly 10% capacity



(a) Histogram (logarithmic x-axis)

(b) Cumulative file sizes (y-axis in log scale)

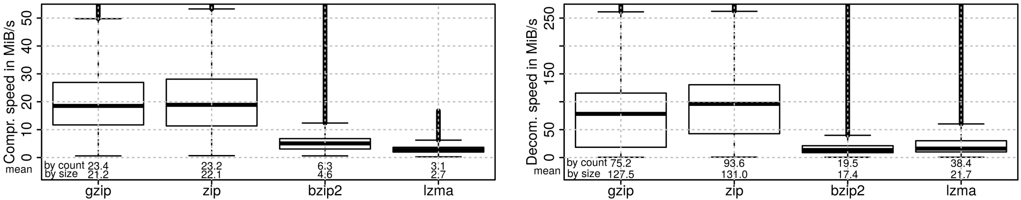
Compression Ratio



Arithmetic mean compression of the full data set for each scientific file format computed on file number. The column “all” shows the mean values for the whole data set. Yellow diamonds show compress % computed by file size

Compression Speed

- Measured user-time for the execution of each tool
  - Ignored I/O
- Again difference between compression by size and count



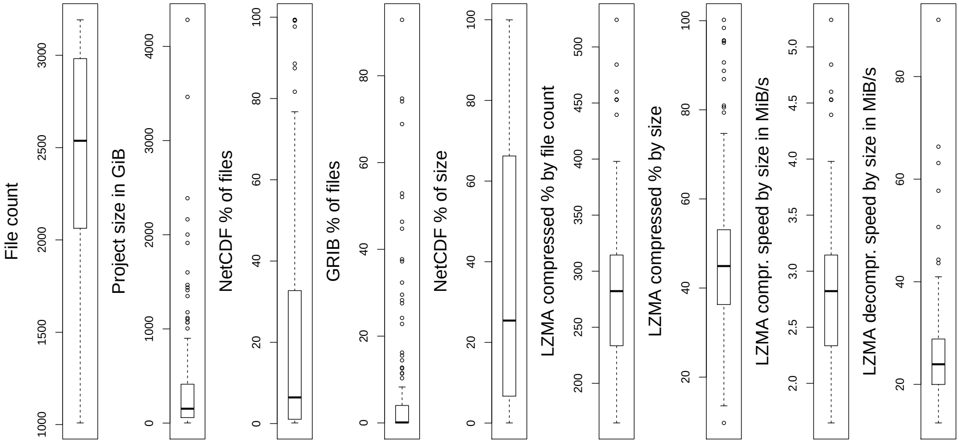
(a) Compression

(b) Decompression

Boxplots showing compression/decompression speed per file, mean shown under the plot

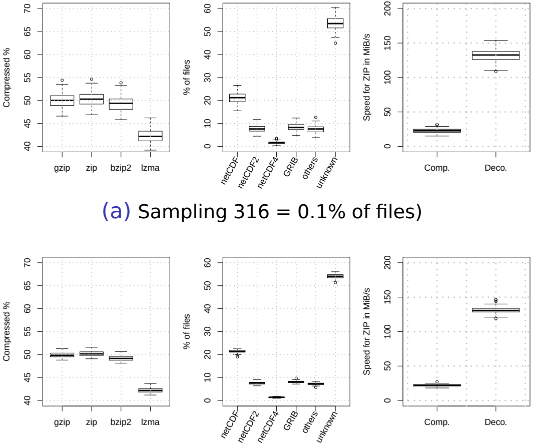
Differences Between Projects

Properties vary significantly → proper sampling requires to pick data from all projects



Analyzing 125 individual projects, each point represents the arithmetic mean value of one

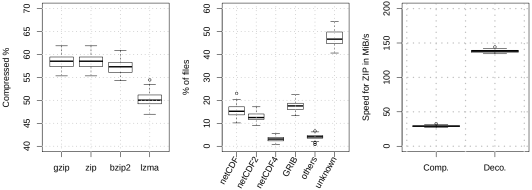
Additional Characteristics Computed by File Count



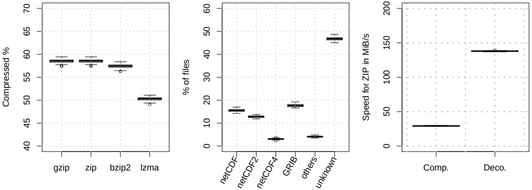
(a) Sampling 316 = 0.1% of files)

(b) Sampling 3164 = 1% of files)

Additional Characteristics Computed by File Size



(c) Drawing 256 samples with replacement



(d) Drawing 4096 samples with replacement