Classifying Temporal Characteristics of Job I/O Using Machine Learning Techniques

Reviews

Eugen Betke
DKRZ
Hamburg, Germany
✉ *betke@dkrz.de*

Julian Kunkel
University of Reading
Reading, UK
✉ *j.m.kunkel@reading.ac.uk*

Abstract

Every day, supercomputers execute 1000s of jobs with different characteristics. Data centers monitor the behavior of jobs to support the users and improve the infrastructure, for instance, by optimizing jobs or by determining guidelines for the next procurement. The classification of jobs into groups that express similar run-time behavior aids this analysis as it reduces the number of representative jobs to look into. This work utilizes machine learning techniques to cluster and classify parallel jobs based on the similarity in their temporal I/O behavior. Our contribution is the qualitative and quantitative evaluation of different I/O characterizations and similarity measurements and the development of a suitable clustering algorithm.

In the evaluation, we explore I/O characteristics from monitoring data of one million parallel jobs and cluster them into groups of similar jobs. Therefore, the time series of various I/O statistics is converted into features using different similarity metrics that customize the classification.

When using general-purpose clustering techniques, suboptimal results are obtained. Additionally, we extract phases of I/O activity from jobs. Finally, we simplify the grouping algorithm in favor of performance. We discuss the impact of these changes on the clustering quality.

Keywords: I/O fingerprinting, performance analysis, monitoring

Keywords: I/O fingerprinting, performance analysis, monitoring

1 Introduction

Scientific large-scale applications of different domains have different needs for I/O and, thus, exhibit a variety of access patterns on storage. Even re-running the same simulation may lead to different behavior. We can distinguish between a temporal behavior, i.e., the operations performed over time such as long read/write phases, bursty I/O pattern, and concurrent metadata operations, and spatial access pattern of individual processes of the application as they can be, e.g., sequential or random.

On different supercomputers, the same I/O patterns may result in different application runtimes depending on the nature of the access pattern. For example, machines equipped with burst buffers [BK18, WOW+14, ?] may significantly reduce application runtimes by absorbing bursty I/O traffic. I/O congestion and file system performance degradation can occur when several I/O intensive jobs are running on the same machine at the same time. I/O aware schedulers, like CARS [LCLA19] and Flux [AGG+14], implement new scheduling strategies that utilize I/O metrics. The analysis of I/O is important not only when I/O begins to take a considerable amount of application runtime but when I/O patterns begin to degrade the performance of the shared file system affecting runtimes of other applications and worsening user experience by unresponsive file systems [KM18].

Understanding the exhibited I/O behavior and implications on the system would give users and administrators information to support analysis by revealing deficiencies and may indicate the potential for I/O optimization. Knowing the potential for optimization is important for the support staff, as it allows them to identify applications that benefit from I/O optimizations. For example, a widely used parallel application that still utilizes sequential I/O might be cost-efficient to optimize.

The main question is how to identify such applications automatically from the observed data. Non-intrusive capturing of I/O metrics and the analysis can be challenging in many aspects. Firstly, in order to find optimization potential, data must be recorded in an appropriate level of detail to retain temporal characteristics. While capturing statistics on node level is supported by many monitoring tools, e.g., LASSi [Kar19], Darshan [Car15], and SIOX [KZH+14]. Detailed metrics on file level are more difficult to obtain. A widespread method is re-implementation and pre-loading of an I/O interface, which contains monitoring code.

However, recording the data isn't enough, the obtained data must be processed but the manual analysis is infeasible as the number of jobs is large – Monitoring systems of HPC systems record data of ten thousand jobs each day. Hence, a semi-automatic approach is required to reduce the number of jobs to investigate.

In different disciplines, machine learning methods have proven to be powerful tools to extract new information from large data sets. Therefore, we explore clustering strategies on monitoring data, to reveal hidden information.

In our previous paper [EB20], we proposed a semi-automatic way to find relevant jobs by computing relevant job characteristics from time series of job behavior. Basically, support staff could then focus on the I/O-intense jobs that express certain metrics the most. Here, we extend the approach by grouping similar jobs based on profiles and I/O-phases in order to simplify the investigation effort. The paper is a significant extension of our previous work in [BK20] where we demonstrated the relevance of utilizing temporal (time series) data in the analysis and applied basic machine learning techniques.

This paper is organized as follows: Section 2 outlines the preliminary work and provides background knowledge that is important to understand the next sections. In Section 3, we discuss the key problem we are dealing with and introduce related work in Section 4. In Section 5, we introduce alternative approaches for the clustering, as different goals for the analysis require different distance metrics, we discuss the variety of approaches. We start with a simple solution and increase complexity. For complex algorithms, we develop examples step by step. After a brief description of the test environment in Section 6.1, we discuss the clustering results of introduced algorithms in Section 6. The discussion includes a use case study of an I/O intensive job. Finally, in Section 7 we summarize the results.

2 Preliminary Work

The German Climate Computing Center (DKRZ) maintains a monitoring system that gathers various statistics from the Mistral HPC system. Mistral has 3,340 compute nodes, 24 login nodes, and two Lustre file systems (lustre01 and lustre02) that provide a capacity of 52 Petabytes. The deployed monitoring system is made up of open source components such as Grafana, OpenTSDB, and Elasticsearch. It includes a lightweight self-developed data collector that captures node statistics continuously – we decided to implement our own lightweight collector since the overhead of existing approaches was much higher. Additionally, the system retrieves various job meta-information from the Slurm workload manager and injects selected log files.

Our motivation for automatic analysis of parallel jobs is the monitoring situation at DKRZ. Mistral runs around 10,000 jobs a day, which is too many for manual analysis. In our previous work [EB20], we found a way to identify I/O intensive jobs and jobs with inefficient usage by deriving statistics from the node-level statistics. This information can aid the procurement of new HPC systems or support the extension of an existing one. While this approach helps to find individual jobs, it doesn't provide a global overview, which might be more important for making decisions at the data center perspective. For example, a discovery of a large group of I/O-intensive and bursty applications would suggest attaching a burst buffer to the storage, to improve application runtimes. In this paper, we take up the idea of I/O categorization from the previous work, where we partition job runtime into equal size segments and map them into three categories (LowIO, HighIO, and CriticalIO), and continue our work to establish a global overview by allowing to classify similar jobs.

Understanding of the following work requires an understanding of data format, that is formed by segmentation and categorization of raw monitoring data.

Raw monitoring data. The monitoring system captures periodically I/O metrics on all client nodes, and sends them to a central database. Figure 2.1 illustrates the structure of the raw monitoring data using an example. In the example, data is captured on two nodes, on two file systems, for two metrics, and at nine time points t_i, resulting in 4-dimensional data (Node × File System × Metric × Time). The number of nodes and the time dimension are variable by the nature of parallel job execution on a cluster. The other dimensions may be fixed for particular HPC systems; here we assume they are variable, to make the approach portable to arbitrary systems. On Mistral data is gathered every five seconds, for two Lustre file system, and for nine I/O metrics [1]. Five of them (md_read, md_mod, md_file_create, md_file_delete, md_other) are aggregates of metadata activities and the remaining four (read_bytes, read_calls, write_bytes, write_calls) capture data access. The md_read covers metadata read-only operations while md_mod covers modifying operations and md_other aggregates rarely used operations such as mknod. The creation and deletion of files are very important and not aggregated.

Segmentation. We split the time series of each I/O metric into equal-sized time intervals (segments) and compute a mean performance for each segment. This stage preserves the performance units (e.g., Op/s, MiB/s) for each I/O metric. The example in Figure 2.1 creates

[1] A difference to previously utilized job statistics is that the Lustre proc files on Mistral doesn't offer Object Storage Client (osc) counters after a major upgrade of Lustre file system from version 2.7 to 2.11. Thus, instead of 13 metrics, this time our data contains only 9 metrics.

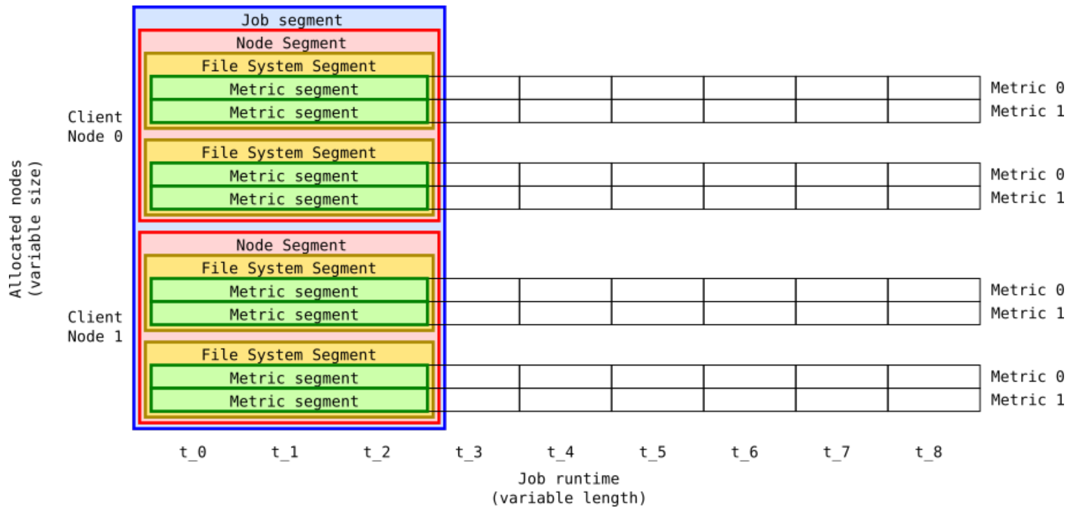


Figure 2.1: An example of 4-dimensional raw monitoring data (Node × File System × Metric × Time) and different levels of segmentation (colored boxes).

segments out of three successive time points just for illustration purposes. Depending on aggregation function, segments can be created of metrics (green boxes), of file systems (yellow boxes), of nodes (red boxes), or even over all dimensions (blue box). The raw monitoring data is converted into a time series of ten minute segments, which we found is a good trade-off to sufficiently represent the temporal behavior of the application while it reduces the size of the time series.

Categorization. Next, to get rid of specific units of individual metrics, and to allow calculations between different I/O metrics, we introduced a categorization pre-processing step that takes into account the performance of the underlying HPC system and assigns a unitless ordered category to each metric segment. For example, how could we compare the metric that reports 10k file opens with another metric that reports 100 MB data access? We use three categories, which are the LowIO=0, HighIO=1 and CriticalIO=4 categories. The category split points are based on the histogram of the indivudual metrics. For any metrics, a segment with a value up to the 99%-Quantile it is considered to be LowIO, larger than the 99.9%-Quantile indicates CriticalIO, and between it is HighIO (see [EB20]). This node-level data can then be used to compute job-statistics by aggregating across dimensions such as time, file systems, and nodes.

In summary, this data representation has the following key advantages for data analysis. The ordered categories make the calculations between different metrics feasible, which is not possible with raw data. Furthermore, the domains are equally scaled and compatible, because the values are between 0 and 4, and a value has a semantical meaning (low, high, or critical IO). Besides, the resulting data representation is much smaller compared to the raw data. This allows us to apply compute-intensive algorithms to large datasets. Finally, as we are mostly interested in jobs with relevant IO, segments mapped to the LowIO category don't distract from significant parts of jobs.

In our previous work, we computed three high-level metrics per job that aid users to understand job profiles:

- **Job-I/O-Balance:** indicates how I/O load is distributed between nodes.
- **Job-I/O-Utilization:** shows the average I/O load during I/O-phases.
- **Job-I/O-Problem-Time** is the fraction of job runtime that is I/O-intensive; it is approximated by the fractions of segments that are considered I/O intensive.

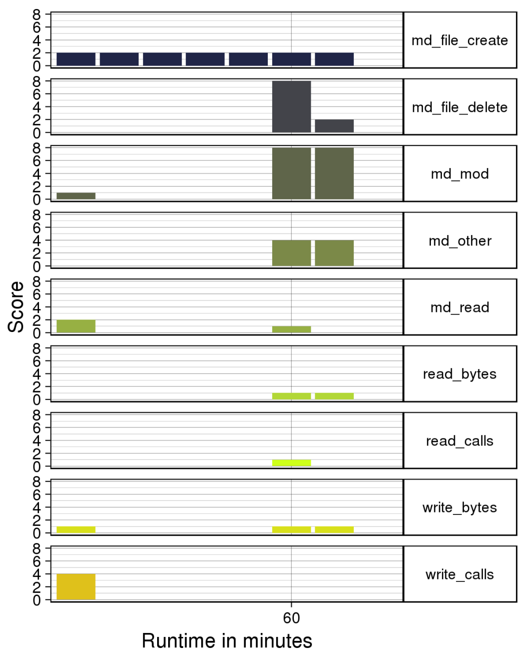


Figure 2.2: Example job-level metric. Score is the sum of all node scores.

In Figure 2.2, we can see the temporal behavior for this particular job when summing up the node-level metrics. We call **an I/O phase a contiguous sequence of non-zero segments** (where the score of the segment is larger than zero). For example, in the figure, we can see one phase for md_file_create, and two phases for md_mod (one short and non-intrusive at the beginning, and one critical around 60 minutes).

When looking at categorization at the metric level, i.e., after reduction of the nodes and file system dimensions, we can observe that many jobs exhibit I/O phases.

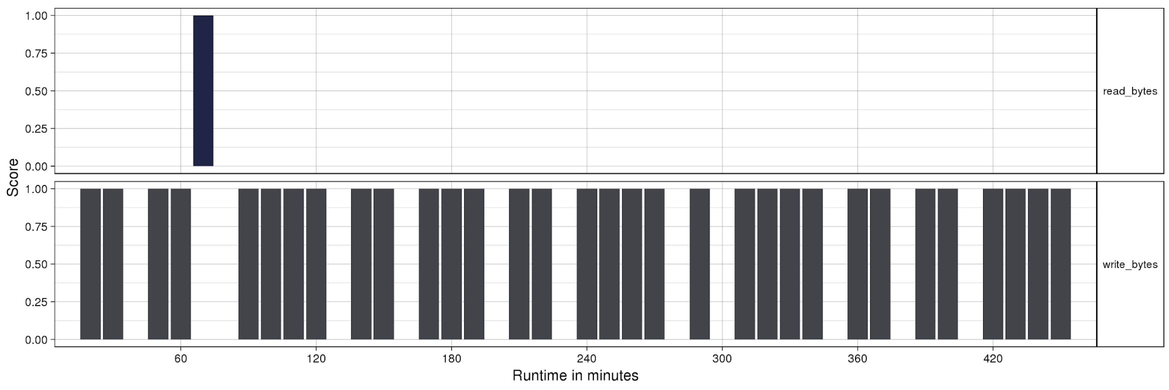
3 Similarity Between Jobs

The raw monitoring data of a job in our environment at DKRZ, we obtain a time series of nine metrics per node, each metric sampled at five seconds intervals. When comparing the time series of such metrics between two jobs, the key question is how do we define the similarity between multiple time series that may even be of different length. ML techniques work well to deal with a fixed number of features, e.g., by creating a fixed-size profile for the jobs and applying dimension reduction techniques such as PCR, we could reduce the complexity and potentially obtain a similar representation for, e.g., two time series.

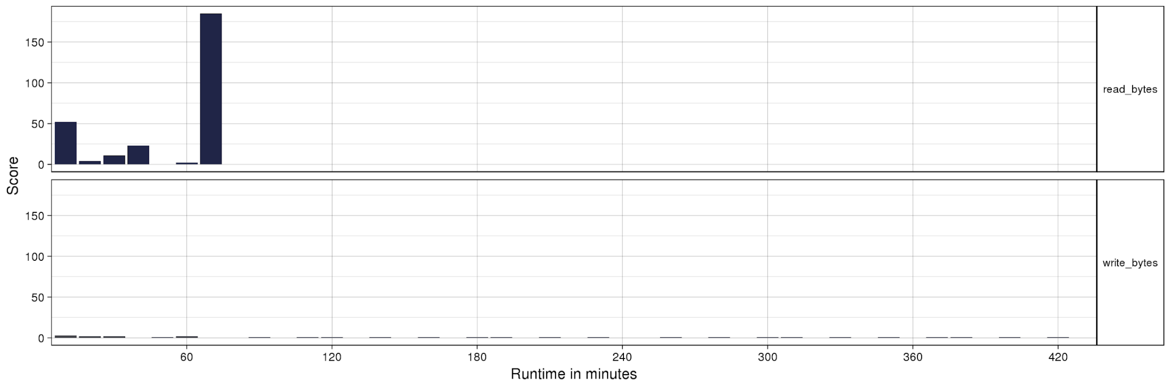
To understand which technique we should use, first, we need to discuss the perception of similarity of I/O patterns from the user perspective. In Figure 3.1, we illustrate the time series of two metrics for three different jobs. The figures show the typical behavior of parallel applications, computation is interrupted by regular I/O phases – by phase, we mean a consecutive segment of time in which certain behavior is exhibited, i.e., the statistics are similarly.

Actually, the shape of I/O phases depends on our definition of I/O phase. By applying dimension reduction techniques, I/O phases from different dimensions can be joined together. Although several aggregations are possible, in this work we focus on I/O phases on metrics, i.e, we aggregate nodes and file system dimensions. When investigating the distribution of I/O phases, we observe that jobs (longer than three segments) exhibit several.

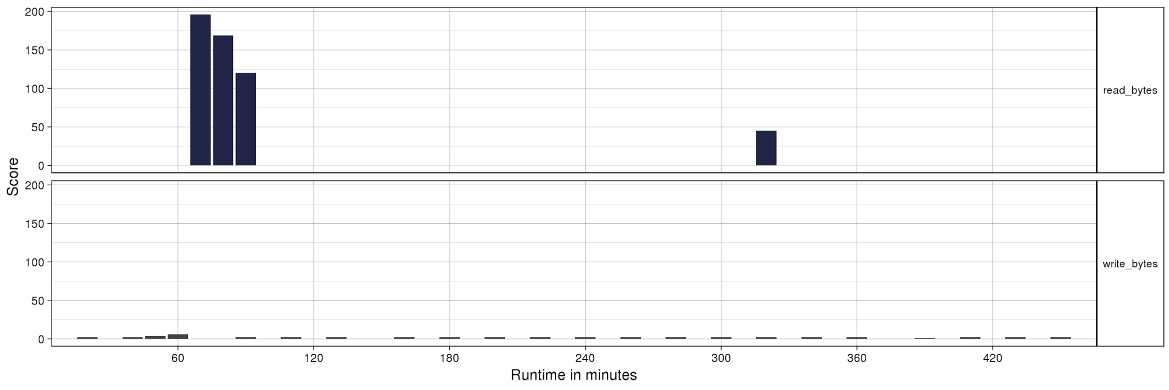
From the user support side, we might be interested in grouping similar suboptimal jobs and aim to provide one recipe to optimize all that exhibit such a behavior. Similarly, we might



(a) Job A runs on 13 nodes



(b) Job B runs on 225 nodes



(c) Job C runs on 40 nodes

Figure 3.1: Monitoring data of three jobs. Score is the sum of individual node scores.

be interested to optimize the pattern for a single I/O phase. Optionally, we may be interested to ignore computation time and focus on I/O phases only. Regardless of the segment of the time series we look at, we naively would consider an I/O pattern to be identical if the time series for all metrics of one job is identical to those of another job. Unfortunately, the obtained measurements vary due to the nature of parallel applications and the environment of the data center they are executed: In practice, different jobs show a different runtime, and even when re-running the same job, the obtained time series varies.

Typical sources of variability when observing performance metrics are: The concurrent usage of the shared storage, a shift between observed I/O phases due to network congestion or CPU throttling for power/heat reasons, and depending on the location of the data in a multi-tier storage system. The same application using a different input configuration may need more compute time, resulting in longer phases between typical I/O patterns; moreover, it may change the I/O pattern. A well-known pattern exhibited by a shorter running application may appear as well in a long-running application. The similarity measure between two jobs should consider those sources of variability and allow us to provide a robust clustering of jobs depending on the user question.

There are different goals for the data analysis done by the user running the application or the support staff of the data center. Whether the timeline is similar when executing an application with a different configuration and input dataset depends on the purpose of the analysis and task that follows the analysis. For these reasons, we decided to explore a variety of alternative options for distance metrics and pre-processing and assess their suitability.

4 Related Work

There are many tracing and profiling tools that are able to record I/O information [KBB+19]; we will discuss a selection of them in more detail in the following. Most of them focus on individual jobs, and only a few of them apply machine learning for data analysis, in particular across jobs. As the purpose of applications is computation and, thus, I/O is just a byproduct, applications often spend less than 10% time with I/O. The issue of performance profiles is that they remove the temporal dimension and make it difficult to identify relevant I/O phases. The Altair tools[2] include Breeze, a user-friendly offline I/O profiling software, an automatic I/O report generator Healthcheck, and command line tool Mistral which purpose is to report on and resolve I/O performance issues when running complex Linux applications on high performance compute clusters. Mistral is a small program that allows you to monitor application I/O patterns in real time, and log undesirable behaviour using rules defined in a configuration file called a contract.

Darshan [CHA+11, Car15] is an open source I/O characterization tool for post-mortem analysis of HPC applications' I/O behavior. Its primary objective is to capture concise but useful information with minimal overhead. Darshan accomplishes this by eschewing end-to-end tracing in favor of compact statistics such as elapsed time, access sizes, access patterns, and file names for each file opened by an application. These statistics are captured in a bounded amount of memory per process as the application executes. When the application shuts down, this data is reduced, compressed, and stored in a unified log file. The Darshan eXtended Tracing (DXT) module can be enabled at runtime to increase fidelity by recording a complete trace of all MPI-I/O and POSIX I/O operations.

This technique allows an application to be traced without modification and with reasonably low overhead. Utilities included with Darshan can then be used to analyze, visualize, and summarize the Darshan log information. Because of Darshan's low overhead, it is suitable for system-wide deployment on large-scale systems. In this deployment model, Darshan can be used not just to investigate the I/O behavior of individual applications but also to capture a broad view of system workloads for use by facility operators and I/O researchers.

There are approaches that monitor record storage behavior and aim to identify inefficient applications in a cluster. TOKIO [LWS+18] integrates logs from various sources to allow an analysis of data. It allows finding certain inefficient access patterns in the data.

The LASSi tool [SRTL19] was developed for detecting, the so called, victim and aggressor applications. An aggressor can steal I/O resources from the victim and negatively affect its runtime. To identify such applications, LASSi calculates metrics from Lustre job-stats and information from the job scheduler. One metric category shows file system load and another category describes applications I/O behavior. The correlation of these metrics can help to identify applications that cause the file system to slow down. In the LASSi workflow this is a manual step, where a support team is involved in the identification of applications during file system slow down. Manual steps are disadvantageous when processing large amounts of data

[2]https://www.altair.com/product-showcase, former Ellexus tools https://www.ellexus.com/products/

and must be avoided in unsupervised I/O behavior identification. LASSi's indicates that the main target group are system maintainers. Understanding LASSi reports may be challenging for ordinary HPC users, who do not have knowledge about the underlying storage system.

In [KB19], the authors utilized probes to detect file system slow-down. A probing tool measures file system response times by periodically sending metadata and read/write requests. An increase of response times correlates to the overloading of the file system. This approach allows the calculation of a slow-down factor identification of the slow-down time period. This approach is able to detect a file system slow-down, but cannot detect the jobs that cause the slow-down.

The NEXTGgenIO monitoring system [EMAM16], has four main collectors. The first two, the PBS and ALPS tools, collect job scheduling information. The proprietary Cray I/O monitoring tool exploits Lustre I/O counters to capture file system usage on OSTs of all OSSs and allocated compute nodes. The MAP component provides information about CPU, memory and network usage. Then all the collected data is clustered for the analysis of system usage and performance evaluation. In contrast to existing approaches, our approach focuses on the analysis of job data and investigates clustering strategy to group similar jobs.

HiperJOBVIZ [NCHD19] is a visual analytic tool for visualizing the resource allocations of data centers for jobs, users, and usage statistics. It provides an overview of the current resource usage and a detailed view of the resource usage via multi-dimensional representation of health metrics. TimeRadar[3] is a part of the tool, which summaries the resource usage via radar charts, creating a kind of comprehensible profile for different user groups.

Trace analysis tools such as Vampir [NAW+96] offer clustering strategies to analyze the per-process time series of a single job, in particular to group similar time series and allow users to focus on the exceptional cases that may lead to slowdown of the parallel application. In [WBH+16], a strategy is developed for determining such a clustering based on execution structure; the paper gives a good introduction for the comparison of two traces as well. The structural clustering algorithm processes function call traces from monitored processes in two stages. First, data reduction is achieved by disregarding time information and mapping function call trees to compact matrix representations. Second, efficient clustering is achieved by using concept lattices [GW99] instead of all-with-all comparison. The authors demonstrate quality improvement of existing analysis techniques, if data is pre-clustering with the introduced structural clustering strategy.

In contrast to existing strategies, we compare a large number of jobs with individual properties.

5 Methodology

The goal of this article is to research the impact of clustering strategies on many jobs from the perspective of a user and data center. Generally, machine learning algorithms expect a fixed number of features, therefore, the time series of statistics that is retrieved on the node-level needs to be pre-processed. The application of a "specific algorithm" can be understood as a number of successive processing steps on data. Roughly speaking, there are three basic steps: data pre-processing (including coding), similarity computation, and clustering. We call one algorithm representing such a combination a *clustering stack*. The pre-processing converts the dynamic-sized monitoring data which depends on the number of captured metrics, allocated nodes, and application runtime into a suitable representation for the clustering algorithm. Then the clustering is applied using a specific similarity function. Finally, the quality of the clustering result is assessed, i.e., how suitable is this strategy for our I/O statistics and use cases?

[3]https://idatavisualizationlab.github.io/HPCC/TimeRadar

Dimension	Operation	Description
Node	Reduce by mean()	Aggregate all nodes by mean() function
	Reduce by sum()	Aggregate all nodes by sum() function
File System	Reduce by mean()	Aggregate all file systems by mean() function
	Reduce by sum()	Aggregate all file systems by sum() function
Metric	Reduce by sum()	Aggregate all nine metrics by sum()
Time		Convert segments to coding formats
All	Reduce by mean()	Reduces time series to a fixed set of values

Table 5.1: Dimension reduction strategies for the original 4D-data.

5.1 Overview

In the following, we have dedicated a section to each step in the analysis. We are listing and briefly discussing potential alternatives. The list of alternatives does not intend to be complete but shows a variety of options.

5.1.1 Data pre-processing

The 4-dimensional data from our monitoring system is too fine-grained for mass analysis. To be able to analyze millions of jobs, we must apply some data reduction techniques. The result of the data-preprocessing is a coding of the initial time series data into one or multiple vectors.

We first convert the time series into segments of ten minutes which we found previously in [EB20] is a good trade-off to represent the temporal behavior of the application while it reduces the size of the time series. Hence, for a job and for each of our nine client-side recorded metrics, we obtain a coarse-grained time series. To simplify the interpretation of results and the choice of the distance metrics, it is beneficial to have the same unit for all features which is why we use our category classification which creates a unitless order. For example, when reduced by node, file system, and across metrics, a point may represent the mean value across all time series for the job for the ten minutes interval.

On the high-level, the following strategies for data reduction could be considered:

- **Job profiles** compute a fixed set of statistics across the whole job regardless of dynamic factors such as runtime or nodes by applying reduction operations such as the arithmetic mean. A drawback of this simplification is that any temporal pattern is lost.
- **Segment statistics** compute for each segment a fixed statistics, such as the mean value across all nodes. This basically preserves the time series and depends on the job-length.
- **Phase statistics** computes high-level statistics by analyzing the temporal behavior of I/O phases further, e.g., computes the average length of IO segments with CriticalIO. This approach is independent of the job length but requires identifying phases and determining meaningful job metrics.

We decided to distinguish the different dimensions of a job (Node, File System, Metric, and Time) defining how the aggregation is performed. Not reducing data in the dimension would mean that the result would depend on, e.g., the number of nodes of a job, which makes the comparison of two jobs difficult. For one dimension, you could compute various statistics; we decided to use mean or sum. Any data reduction is performed in this order across the dimensions: first File System, Node, and last by Metric. Thus, a time series of encoded segments remain. The time series of segments can be reduced differently in each dimension; alternatives that we apply in this paper are listed in Table 5.1. By reducing along all dimensions, you obtain basically a job profile (as discussed above).

Operation	Description
Binarization	Segments are mapped to 9-bit numbers (v), where each position (i) represents a metric. The bits are set by the following function: $$v_i = \begin{cases} \text{false if segment}_i = 0 \\ \text{true otherwise.} \end{cases} , i \in [\text{enumerated metrics}] \quad (1)$$
Quantization	Quantize segments to NonIO + 16 I/O levels.
Zeros-Aggregation	Merge all consecutive zero segments.
Phase-Extraction	Extract continuous sequences of non-zero segments. - Split time series at zero segments in sub time series (I/O phases) and remove zero segments. - Preserve order of I/O phases.

Table 5.2: Coding operations.

5.1.2 Coding

Segmented data contains a numeric floating-point value for each data, which can be still too much information for the analysis. Further, we need appropriate data representation for similarity computation between time series. Therefore, we introduce two condensed data representations called binary and quantum coding. Additionally, we introduce two operations on the coding: the first extracts I/O phases – a phase is a consecutive non-zero series of segments; the second merges all consecutive zero segments into one segment, i.e., removes compute phases in the statistics. The operations are summarized in Table 5.2 and described in the following.

B-coding Binary coding represents monitoring data as a sequence of numbers, where each number represents the overall file system usage. The number is computed based on the nine metrics found in the segment, e.g., if a phase is read-intense and write-intense, it is encoded as one type of behavior. In this approach, each conceivable combination of activities has a unique number.

The approach maps the three categories to the following two states: The LowIO category is mapped to the non-active (0) state, and HighIO and CriticalIO categories are mapped to the active (1) state. On one side, by doing this, we lose information about performance intensity, but on other side, this simplification allows a more comprehensible comparison of job activities.

In our implementation, we use a 9-bit number to represent each segment, where each bit represents a metric. The bit is 1 if the corresponding metric is active, and 0 if not. Translated to the decimal representation, metric segments can be coded as 1, 2, 4, 8, 16, and so on. Using this kind of coding we can compute a number for each segment, that describes unambiguously the file system usage, e.g., a situation where intensive usage of md_read (Code=16) and read_bytes (Code=32) occur at the same time and no other significant loads are registered is coded by the value 48. Coding is reversible, e.g., when having value 48, the computation of active metrics is straightforward.

To reduce the 4-dimensional data, we reduce that structure to two dimensions (segments metrics) by aggregating other dimensions by applying sum() function on score values. In the resulting table we leave zero scores, and change scores larger than zero to one. After coding each segment, the jobs can be represented as a sequence of numbers, e.g.,

```
[1:5:-:-:-:-:-:-:-:96:96:96:96:96:96:96]
```

Listing 5.1: B-coding of a 15 segments long job.

The monitoring dataset doesn't provide information about what happens during the zero segments, it could be idle time, computational work, or other non-I/O operations. It can also be that the job script cannot start immediately or run on a slow network. To catch such

Distance	Formula	Domain	Description
Euclidean	$d_e = \sqrt{(\sum_i^n (a_i - b_i))}$	$a, b \in \mathbb{R}^n$	The Euclidean distance between two vectors.
Levenshtein	$d_{\text{lev}} = \text{lev}(a, b)$	$a, b \in$ Time Series	Number of operation (inserts, deletes, and changes) requires to convert one coding in another.
Normalized-Difference	$d_{\text{nd}} = \frac{\lvert a-b \rvert}{16}$	$0 \leq a, b \leq 16$	A change from level 2 to level 6 on a 16 level system would be (6 - 2) / 16 = 0.25.
Relative-Change	$d_{\text{rc}} = \frac{\min(a,b)}{\max(a,b)}$	$a, b \in \mathbb{R}^+$	A change from level 2 to level 6 system would be 2/6 = 0.66.

Table 5.3: Selection of distance metrics that can be used to compute similarities.

jobs, we aggregate multiple consecutive zero segments into one zero segment, thus the coding of the previous job would replace all 0:...:0 sequences just with one 0.

```
[1:5:-:96:96:96:96:96:96:96]
```

Listing 5.2: B-coding of a 15 segments long job with zero aggregation applied.

Note, that the actual job length is kept for analysis purposes, thus, only the length of the encoded sequence changes.

Q-coding Quantum coding preserves monitoring data for each metric and each segment. As the name suggests, the value of a segment is quantized into a number to allow creation of a string representing the I/O behavior. We use uniform quantization to convert monitoring data to time series of 16 levels, [0,0.25) → 0, [0.25, 0.5) → 1, . . . , [3.75,4] → F.

The example below shows Q-coding for a job containing 6 segments.

```
'md_file_create'  :  [-:-:2:2:2:9]
'md_file_delete'  :  [-:-:-:-:-:-]
'md_mod'          :  [-:-:-:-:-:-]
'md_other'        :  [-:-:-:-:-:-]
'md_read'         :  [-:-:-:9:3:-]
'read_bytes'      :  [5:-:-:-:-:-]
'read_calls'      :  [-:-:-:-:-:-]
'write_bytes'     :  [-:-:-:-:9:9]
'write_calls'     :  [-:-:-:-:-:-]
```

Listing 5.3: Q-coding of a six segments long job.

5.1.3 Similarity Metrics

Determining the resemblance between two jobs is the main task of this work. Clustering algorithms need distance metrics to enable construction of the clusters from individual components. There are various variants, a selection is listed in Table 5.3. The Euclidean distance for two vectors can naturally only be applied to vectors of the same length. The Levenshtein distance is the number of modifications (inserts, deletes, changes) of individual characters that need to be made to transform one string to another. It can also be applied on other data types such as vectors and time series.

The Normalized-Difference and Relative-Change distances work with single values, and can not be applied directly on time series. To make them work with time series we use the following approach. Firstly, we introduce in Table 5.4 two additional operations that transform time series to the same length. The first operation, Trimming, can remove segments at the beginning, or/and at the end of the longer job. The second operation, Zeros-Insertions, can insert zeros (LowIO segments) at the beginning, between, or/and at the end of I/O phases of both jobs. Both operations create equal sized time series with the highest possible similarity. Finally, we use a Normalized-Difference-based or Relative-Change-based similarity functions to compute similarity between segments, and obtain the overall similarity by computing the mean.

Operation	Description
Trimming	Remove segments at the beginning or/and at the end of the longest time series.
Zeros-Insertion	Insert zero-segments at the beginning, between I/O phases, or/and at the end of time series.

Table 5.4: Potential length adjustment strategies create time series with the same length.

Algorithm	Description
k-means	Runs a random initialization, then refines the clusters until the clusters don't change further.
Density clustering	Group nearby elements recursively together as long as they are (ϵ) close together.
Agglomerative	Hierarchically cluster data into groups of elements that belong closer together.
Clustering-Tree	Create a hierarchical clustering for subset of data to train a decision tree, then apply the decision tree on the full dataset.
Simplified-Density	Combines ideas from k-means with density clustering. The simplified clustering algorithms add a job to a cluster if similarity to the cluster centroid is equal or larger than the user defined similarity (SIM) value.

Table 5.5: Selection of clustering algorithms

5.1.4 Clustering

In the last step, similar jobs need to be grouped in clusters. A subset of potential strategies is listed in Table 5.5. Often, the k-means algorithm is applied which utilizes a random strategy to initialize clusters. However, as it requires us to define a fixed number of clusters, it is problematic as the number of actual clusters are unknown. The large class of density clustering algorithms such as DBSCAN groups objects that are "close" together into one class; a user defines the maximum distance of objects that belong to the same class by setting an ϵ which represents the similarity value. Agglomerative strategies group similar elements together to form a hierarchy of some sort, e.g., a tree. One of the challenges for algorithms like this is that they have a computational complexity that is quadratic or greater, meaning the computational cost of their implementations increases very quickly with increases in the size of the input dataset. One of the requirements we have for the techniques we are defining is that it should be able to complete the analysis in a reasonable amount of time.

We developed two strategies that address the performance issue and the other limitations, the Clustering-Tree algorithm and the SimplifiedDensity algorithm. These algorithms are able to handle the large number of jobs but aim to preserve the core ideas of k-means/density algorithms. They are described in the following:

Clustering-Tree algorithm This algorithm involves three steps:

1. Agglomerative clustering of a small dataset, enumerate similar leaf-level clusters.
2. Training of a decision tree model on the small dataset, i.e., the tree should decide in which cluster a job is stored.
3. Clustering of the large data set using the decision tree.

Since the number of clusters on leaf-level depends on their similarity, this strategy should determine a number of suitable clusters that mimics the job distribution in the training set.

Simplified-Density algorithm The algorithm forms clusters around centroids. These are job codings that form clusters by attracting similar jobs. All jobs in a cluster fulfill only one condition, the similarity (SIM) to the centroid has to be bigger than the user defined minimum, in that sense it shares some similarity to DBSCAN. It is approximative as the order of jobs matters for the creation of the clusters.

The algorithm can be summarized as follows:

1. Pick one arbitrary job from the dataset, which will serve as a cluster centroid, and compute similarity to all other jobs.
2. Create a cluster of jobs that have a minimum similarity and remove them from the dataset.
3. Repeat step 1 and 2 until no jobs are left in the dataset.

This approach allows a performant implementation, but they also allow a job to be in one cluster, even if a centroid of another cluster is much closer, as long as the condition is true. Hence, the result depends on the processing order of the jobs, though it is deterministic.

5.1.5 Clustering Stacks

There are various combinations of the different strategies possible. For simplicity, we refer to one clustering stack just as an algorithm. During our research, we explored various combinations out of the possible combinations, but we found that these do not perform well and are not interesting to discuss. We don't mention them in the paper. The operation stack which we describe in this article is visualized in Figure 5.1. In brief, the algorithms can be characterized like in Table 5.6.

Coding

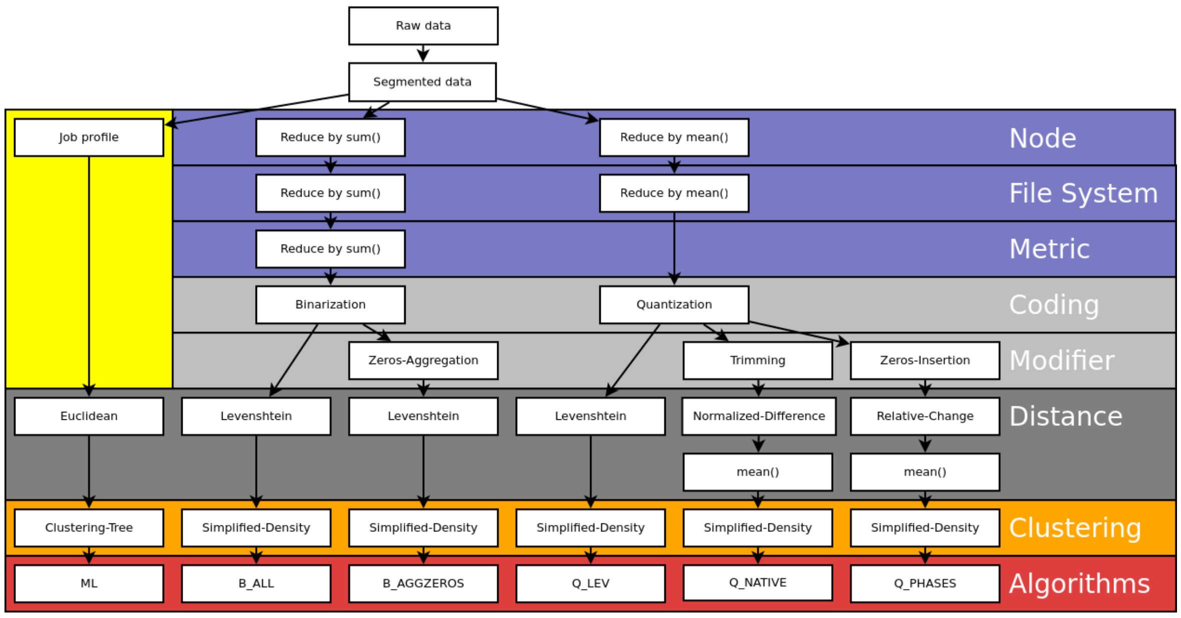


Figure 5.1: Algorithms and their actual operation stacks.

Algorithm	Characterization
ML	a set of **traditional machine learning** techniques on job profiles.
B_ALL	**Levenshtein** distance applied on **B-codings.**
B_AGGZEROS	**Levenshtein** distance applied on **B-codings**, but with **zero aggregation** of contiguous zero segments.
Q_LEV	**Levenshtein** distance applied on **Q-coding**.
Q_NATIVE	A **performance-aware** algorithm operating on Q-coding.
Q_PHASES	A **performance-aware** and **I/O phase-aware** algorithm.

Table 5.6: Algorithm characterization

5.2 Algorithms

This section describes the evaluated algorithms in detail.

5.2.1 ML

To apply existing clustering algorithms, first, a job-profile is created in the pre-processing. The 4d time series can be transformed into the required fixed-size input format accepted by the general-purpose ML clustering algorithms. In the pre-processing step, the MinMaxScaler scales the features to values between 0 and 1 using MinMax normalization. The Euclidean distance is used; therefore, the highest distance between two points can be at most $\epsilon_{\max} = d^{1/d}$, where d is the dimension of the dataset.

We explored two job profiles: IO-metric and IO-duration. The **IO-metric job profile** utilizes three features, Job-IO-Balance, Job-IO-Utilization, and Job-IO-Problem-Time (as defined in [EB20]). After the data pre-processing, we obtain a set of 3-dimensional data points with a domain between 0 and 1. The maximum distance between any two jobs (ϵ_{max}) is 1.44.

The **IO-duration job profile** contains the fraction of runtime, a job spent doing the individual I/O categories leading to 27 columns. The columns are named according to the following scheme: metric_category, e.g, bytes_read_0 or md_file_delete_4. The first part is the one of the nine metric names and the second part is the category number (LowIO=0, HighIO=1 and CriticalIO=4). These columns are used for machine learning as input features. There is a constraint for each metric (metric_0 + metric_1 + metric_4 = 1), which we can use to reduce the number of features by 9, because one feature in each metric is redundant. Consider an example where 20% of application runtime has low write performance (write_bytes_0 = 0.2) and 70% of application runtime the write performance is high (write_bytes_1=0.7). Then, we can conclude that the remaining 10% of write performance was critical (write_bytes_4 = 0.1). A similar computation we can do for other metrics, which makes 9 features redundant, because they can be computed from the other features. So we have to deal with 18 features; ϵ_{max} is 1.17.

Clustering algorithm We found that the size of our datasets is the one of the obstacles to selecting and using the suitable clustering algorithms. For example, the scaling behavior of the agglomerative clustering algorithm that is used in this work can handle around 10,000 jobs in a reasonable amount of time as the complexity is at least $O(N^2)$.

The agglomerative clustering algorithm uses Euclidean distance as a similarity measure, i.e., if the component wise distance between two job vectors is shorter than the defined distance (ϵ), then these jobs belong to the same cluster.

Therefore, to be able to cluster 1,000,000 samples, we created a variant of the algorithm by utilizing a decision tree. This workaround involves three steps:

1. Cluster and label 10,000 jobs with the agglomerative clustering algorithm
2. Train a decision tree model with data from the previous step

3. Predict labels of 1,000,000 jobs with the trained decision tree model

5.2.2 B_ALL

This algorithm converts the time series data via B-coding to a string and then determines the relative similarity using the Levenshtein distance. The Levenshtein based similarity between two jobs is determined using Equation (2).

$$\text{similarity}\left(\text{job}_A, \text{job}_B\right) = 1 - \frac{\text{levenshtein}\left(\text{job}_A, \text{job}_B\right)}{\max\left(|job_A|, |job_B|\right)} \tag{2}$$

It is the number of Levenshtein operations (changes/deletes/inserts) divided by the length of the longest sequence, and subtracted from the value one.

```
[1:5:-:-:-:-:-:96:96:96:96:96:96:96]
```

(a) Job A

```
[-:-:-:-:-:-:-:-:96:96:96:96:96:98]
```

(b) Job B

Listing 5.1: B_ALL: The similarity between these two jobs is 73 percent

5.2.3 B_AGGZEROS

This is a variant of B_ALL where subsequent zero-sequences are reduced to a single zero segment before Equation (3) is applied. This allows us to focus on I/O intensive parts of the job. The example below shows the reduced codings from the previous example.

$$\text{similarity}\left(\text{job}_A, \text{job}_B\right) = 1 - \frac{\text{levenshtein}\left(\text{job}_A, \text{job}_B\right)}{\max\left(|job_A|, |job_B|\right)} \tag{3}$$

```
[1:5:-:96:96:96:96:96:96:96]
```

(a) Job A

```
[-:96:96:96:96:96:98]
```

(b) Job B

Listing 5.2: B_AGGZEROS: The similarity between these two jobs is 53 percent

5.2.4 Q_LEV

This algorithm works on the same principle as the B algorithms, with the difference that it applies a Q-coding and then computes the mean similarity across all metrics using Levenshtein. This adaption allows a comparison of time series where one metric differs as only one metric string would differ while with binary coding the overall string would be different.

$$\text{similarity}\left(\text{job}_A, \text{job}_B\right) = 1 - \frac{\sum_{m \in Metrics} \text{levenshtein}\left(\text{job}_{A,m}, \text{job}_{B,m}\right)}{|\text{Metrics}| \cdot \max(|job_A|, |job_B|)} \tag{4}$$

5.2.5 Q_NATIVE

This algorithm computes a performance-aware similarity between two jobs. It works with a selection of I/O intensive metrics from Q-codings ($\text{Metrics}_{\text{IO}}$) and utilizes the Normalized-Difference distance for similarity computation. Assume there are two jobs, job_A and job_B. I/O intensive metrics are those, which have either in job_A or in job_B at least one non-zero segment. Further, assume job_B is larger or equal than job_A. Then the job similarity between the two jobs can be computed using Equation (5).

$$\text{similarity}\left(\text{job}_A, \text{job}_B\right) = \frac{\sum_{m \in \text{Metrics}_{\text{IO}}} \text{maxSim}\left(\text{job}_{A,m}, \text{job}_{B,m}\right)}{|\text{Metrics}_{\text{IO}}|} \tag{5}$$

The algorithm assumes that the metric codings $job_{B,m}$ and $job_{A,m}$ can be of different lengths. Therefore, to compute similarity, it trims the most irrelevant segments of the longer time series, which are determined by the sliding window approach, with the goal to find maximum similarity between two metrics. That means, we take the shortest coding sequence (CA) and check how it fits best on the longer coding sequence (CB). The best fitted slice is denoted as (SB).

```
float maxSim(CA, CB) {
  max = 0
  LA = length(CA)
  LB = length(CB)
  for pos in 0..(LB-LA) { // LB-LA is the boundary, as stated: LB > LA
    SB = substr(str=CB, start=pos, len=LA)
    sim = sliceSim(CA, SB)
    if sim > max {
      max = sim
    }
  }
  return max
}
```

Listing 5.4: Pseudo code of the maxSim() function

The `sliceSim()` in Equation (6) computes similarity between two codings of the same length. It iterates over all positions of the coding C_A and slice S_B, computes segment similarities, and the sum. Lastly, it normalizes the sum by the length of job_B. This function maps the differences between performance levels of the same segments (values between 0 and 16) to a relative value between 0 and 1.

$$\text{sliceSim}\left(C_A, S_B\right) = \frac{1}{|job_B|} \sum_{\text{pos}=1}^{|job_A|} \left(1 - \frac{|C_{A,\text{pos}} - S_{B,\text{pos}}|}{16}\right), \text{ with } |job_B| \geq |job_A| \tag{6}$$

For illustration, we apply the approach to compute similarity between two jobs listed in Listing 5.3. First, we determine and trim the I/O intensive metrics and use the Equation (6) to compute individual metrics similarities. The results are listed in Table 5.7. Then, we compute the mean in Table 5.8. According to this algorithm, the two jobs in the table are similar to 73% as the performance values of the I/O metrics are actually close to each other.

```
md_file_create [-:-:-:-]
md_file_delete [-:-:-:-]
md_mod         [-:-:-:-]
md_other       [-:-:-:-]
md_read        [1:1:-:-]
read_bytes     [1:1:2:2]
read_calls     [-:-:-:-]
write_bytes    [-:-:-:-]
write_calls    [-:-:-:-]
```

(a) Coding of Job A

```
md_file_create [-:-:-:-:-]
md_file_delete [-:-:-:-:-]
md_mod         [-:-:-:-:-]
md_other       [-:-:-:-:-]
md_read        [5:5:-:-:-]
read_bytes     [1:1:1:2:2]
read_calls     [-:-:-:-:-]
write_bytes    [-:-:4:8:-]
write_calls    [-:-:-:-:-]
```

(b) Coding of Job B

Listing 5.3: Q_NATIVE: Q-codings of two jobs and I/O intensive metrics.

Metric_{IO}	C_A	C_B	pos	$S_{B,pos}$	sliceSim(C_A, S_B)
md_read	[1:1:-:-]	[5:5:-:-:-]	0	[5:5:-:-]	0.7
md_read	[1:1:-:-]	[5:5:-:-:-]	1	[5:-:-:-]	0.7375
read_bytes	[1:1:2:2]	[1:1:1:2:2]	0	[1:1:1:2]	0.7875
read_bytes	[1:1:2:2]	[1:1:1:2:2]	1	[1:1:2:2]	0.8
write_bytes	[-:-:-:-]	[-:-:4:8:-]	0	[-:-:4:8]	0.65
write_bytes	[-:-:-:-]	[-:-:4:8:-]	1	[-:4:8:-]	0.65

Table 5.7: Q_NATIVE: Slice similarities $S_{B,pos} = \text{substr}(\text{str} = C_B, \text{start} = pos, \text{len} = L_A)$

Metric_{IO}	C_A	C_B	maxSim(C_A, C_B)
md_read	[1:1:-:-]	[5:5:-:-:-]	0.7375
read_bytes	[1:1:2:2]	[1:1:1:2:2]	0.8
write_bytes	[-:-:-:-]	[-:-:4:8:-]	0.65
Mean()			Similarity ≈ 0.73

Table 5.8: Q_NATIVE: Similarity computation.

5.2.6 Q_PHASES

The developed phase-matching quantization algorithm allows us to cluster variable length jobs by aligning IO phases. First, using a Q-coding, phases of consecutive I/O segments are detected and extracted. Then, the best match between the (disjoint) phases is determined, finally the similarity is computed. We will discuss the details of each step on the example jobs in Listing 5.4.

```
md_file_create  [-:-:2:2:2:9:3:-:9:1:1:1:-]
md_file_delete  [-:-:-:-:-:-:-:-:-:-:-:-:-]
md_mod          [-:-:-:-:-:-:-:-:-:-:-:-:-]
md_other        [-:-:-:-:-:-:-:-:-:-:-:-:-]
md_read         [-:-:-:-:-:-:-:-:-:-:-:-:-]
read_bytes      [-:-:-:9:3:-:-:-:-:-:1:-:-]
read_calls      [-:-:-:-:-:-:-:-:-:-:-:-:-]
write_bytes     [-:-:-:-:-:-:-:-:-:-:-:-:-]
write_calls     [-:-:-:-:-:-:-:-:-:-:-:-:-]
```

(a) Coding of Job A

```
[[2,2,2,9,3], [9,1,1,1]]
[]
[]
[]
[]
[[9,3], [1]]
[]
[]
[]
```

(b) I/O phases of Job A

```
md_file_create  [1:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-]
md_file_delete  [-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-]
md_mod          [-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-]
md_other        [-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-]
md_read         [-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-]
read_bytes      [2:2:2:2:8:2:2:-:-:-:1:-:-:8:1:1]
read_calls      [-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-]
write_bytes     [-:-:-:-:-:-:-:-:-:-:-:5:-:-:-:-]
write_calls     [-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-]
```

(c) Coding of Job B

```
[[1]]
[]
[]
[]
[]
[[2,2,2,2,8,2,2], [1], [8,1,1]]
[]
[[5]]
[]
```

(d) I/O phases of Job B

Listing 5.4: Q_PHASES: Q-codings of two jobs and their I/O phases.

Phase detection According to our I/O phase definition, phases are separated by zeros. This definition makes the detection of individual phases to a trivial task. Relevant for similarity computation are only metrics that contain at least one non-zero segment. In this example, that are the md_file_create, read_bytes and write_bytes metrics.

Phase matching In Listing 5.5 we slide the shorter phase over the longer and compute similarity to the corresponding part such that we can find the best match. In this example, the best match is found on position four.

After repeating this step for all possible combinations of I/O phases, we chose those with the pairs with the highest similarity, and computed the match between them. They are listed

```
[2:2:2:2:8:2:2]
[9:3:-:-:-:-:-]
similarity
= (2/9+2/3)/7
= 13%
```

(a) Shift 0 positions

```
[2:2:2:2:8:2:2]
[-:9:3:-:-:-:-]
similarity
= (2/9+2/3)/7
= 13%
```

(b) Shift 1 position

```
[2:2:2:2:8:2:2]
[-:-:9:3:-:-:-]
similarity
= (2/9+2/3)/7
= 13%
```

(c) Shift 2 positions

```
[2:2:2:2:8:2:2]
[-:-:-:9:3:-:-]
similarity
= (2/9+3/8)/7
= 8%
```

(d) Shift 3 positions

```
[2:2:2:2:8:2:2]
[-:-:-:-:9:3:-]
similarity
= (8/9+2/3)/7
= 22%
```

(e) Shift 4 positions

```
[2:2:2:2:8:2:2]
[-:-:-:-:-:9:3]
similarity
= (2/9+2/3)/7
= 13%
```

(f) Shift 5 positions

Listing 5.5: Phase matching example.

Metric	I/O Phase A	I/O Phase B	Match	Length
md_file_create	[-:-:-:-:-]	[2:2:2:9:3]	0	5
	[-:1:-:-]	[9:1:1:1]	1	4
read_bytes	[2:2:2:2:8:2:2]	[-:-:-:-:9:3:-]	8/9 + 2/3	7
	[1]	[1]	1	1
	[8:1:1]	[-:-:-]	0	3
write_bytes	[5]	[-]	0	1
Sum			3.56	21

Table 5.9: Best combination of I/O phases, that achieves the highest possible similarity.

in Table 5.9. I/O phases without a pair are assigned virtually to empty phases, and their match is zero. To compute the match we divide the lowest value for each segment and sum up the values. The example below shows the pairs of I/O phases and their matches.

Similarity Next, the similarity for coding with different lengths must be adjusted. Similarity between two jobs is the sum of matches divided by the sum of lengths of the longer phases. The calculation is done in Equation (7). In this example, we get a similarity of 17%.

$$\text{similarity}(\text{job}_A, \text{job}_B) = \frac{\sum_{m,i} \text{match}_{m,i}}{\sum_{m,i} \text{length}_{m,i}} = \frac{3.56}{21} \approx 0.17, \text{ with } m \in \text{Metrics}, i \in \text{Phases of } m \tag{7}$$

5.3 Assessment

Lastly, the quality of the obtained clusters must be assessed. Overall, we will assess their suitability using quantitative metrics such as the number of generated clusters and their sizes and qualitatively by manually exploring clusters of relevant jobs. We want to emphasize that our goal is to find similar jobs. Depending on data, the clustering algorithms may create thousands of clusters. Unfortunately, it is not feasible to analyze all of them qualitatively with reasonable effort and there are no tools that can assess the cluster quality automatically.

Therefore, for the assessment, we look inside clusters and inspect segment sequences of the corresponding jobs. For the quantitative analysis, we attempted to find clusters that show the characteristic weakness of the algorithm and discuss them. We also define *cluster relevance* as a metric that assists in determining which cluster bears the best optimization potential and, hence, could be investigated first by the support staff.

For the qualitative analysis, we start by looking into a job that is given to user support, then similar jobs need to be found. In the same cluster, we expect the sequences to be similar. If not, the clustering algorithm is not effective.

Algorithm	Number of I/O intensive jobs
B_*, Q_NATIVE	583.000
Q_LEV, Q_PHASES	395.000

Table 6.1: Numbers of I/O-intensive jobs.

6 Evaluation

6.1 Test Environment

For the performance tests, we allocate a compute node on Mistral supercomputer. It is equipped with 2x Intel® Xeon® CPU E5 2680 v3 @ 2.50GHz, 64GB DDR4 RAM. For clustering of job profiles, we use the agglomerative clustering algorithm, decision trees, and the MinMaxScaler from the Scikit-Learn 0.22.1 library and Python 3.8.0. For clustering of B-codings and Q-codings we use a clustering algorithm implemented in Rust 1.42.0, and run it on a single core.

6.2 Description of Our Data Set

This section describes the job data extracted from Mistral, originally we gathered 1 million jobs from a period of 203 days. Mostly, jobs are allowed to run up to 8-hours, leading to time series with up to 48 segments. From the perspective of this work, analysis of non-I/O-intensive jobs (jobs with zero in all segments) is irrelevant, these jobs can be grouped into one class easily. For that reason, we detect zero-jobs early and remove them from the dataset; these are about 40% of jobs.

The number of zero-jobs is different for Q-codings and B-codings. The reason is the quantization to Q-coding which computes mean performance values for all segments and then quantizes them into NonIO + 16 levels. Hereby, some segments can be quantized to zeros, if the mean value is lower than 0.125. Therefore, it may happen that some jobs fall into the zero-job category, if all segments are quantized to zeros. It can not happen in B-coding, because it preserves all active segments. Interestingly, it affects around 14% of jobs. Table 6.1 shows the number of jobs that are analyzed further.

6.2.1 Data exploration

I/O phases are, according to our definition, contiguous sequences of I/O-intensive segments. The statistics of the number phases for each different metric are visualized in Figure 6.1. These histograms show for one metric how many I/O phases a job has. Most jobs exhibit up to 5-phases in one metric. The metrics (md_bytes, md_mod, and md_file_delete) have the longest tail; some job use up to 22-phases, meaning that almost every other segment the value is zero or ≥ 1.

6.3 Explored Algorithm Parameters

ML We explored our discussed job profiles: IO-metric and IO-duration. For both datasets we explore $\epsilon \in [0.03, 0.06, 0.09, 0.1, 0.2, 0.3]$.

B/Q-algorithms In the experiments with B/Q-algorithms, we cluster jobs varying the similarity parameter SIM $\in [0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99]$. Additionally, we capture the clustering progress each time after clustering 10,000 jobs.

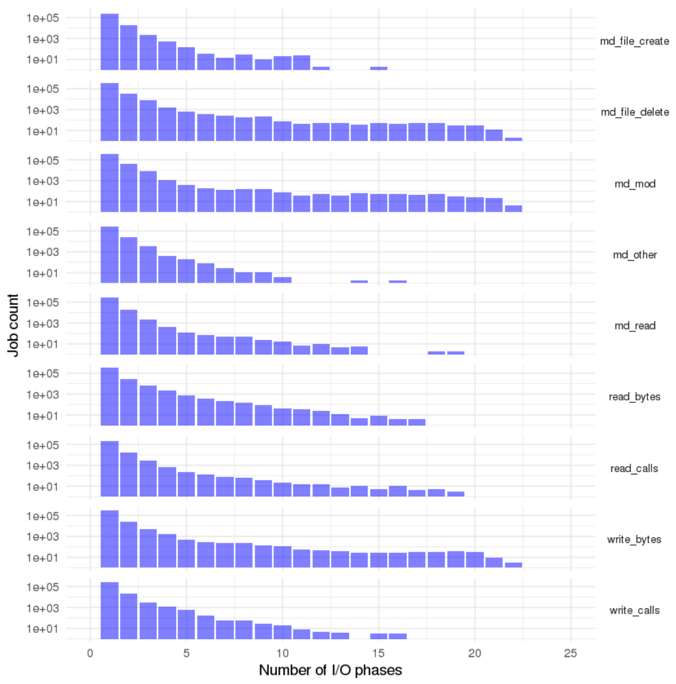


Figure 6.1: Histogram showing the number of I/O phases for all metrics.

6.4 Limitations of the ML Algorithms on Job-Profiles

This section contains data statistics, and the clustering results in Figure 6.2 for applying the ML algorithm (hierarchical clustering + decision tree) but using the two different job profiles (IO-Metric and IO-Duration). Due to lack of tools, we determine cluster quality on a small scale.

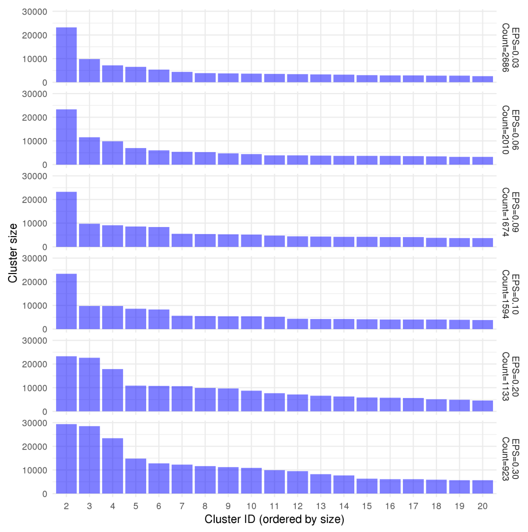
6.4.1 Results

We conducted several experiments with different ϵ values between 0 and ϵ_{max}. The Top 20 largest clusters and total number of clusters is visualized in Figure 6.2a and Figure 6.2b for IO-metric and IO-duration job-profiles, respectively.

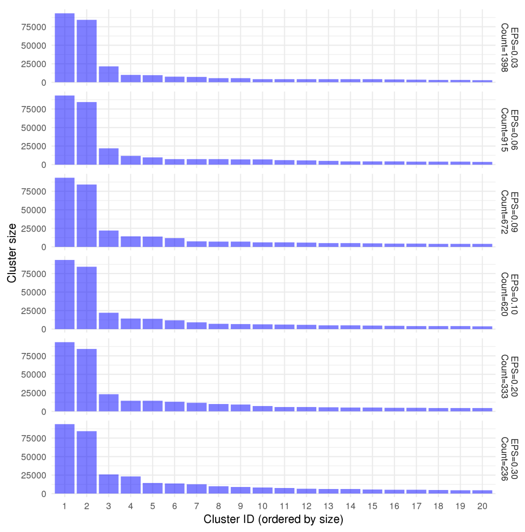
We can see that IO-duration generated two very large clusters regardless of ϵ. From Cluster 3 of IO-duration, the number of jobs inside looks similar as for IO-metric. In IO-metric, we can see that with increasing ϵ, the number of jobs per cluster increases (as expected).

A look inside clusters reveals chaotic clustering results. For both datasets and for all ϵ values we could find many coding sequences in a cluster that we wouldn't locate together – while the job profiles are similar, the timelines are absolutely different. The examples in Table 6.2 show some jobs in one cluster for a high similarity ($\epsilon = 0.03$).

We looked in the Top 10 largest clusters and a random selection of clusters to find the same picture: Even with low ϵ values the algorithms appear to produce polluted clusters, i.e, with samples from other clusters. Generally, the algorithm does the intended job – clustering jobs with similar profiles together. However, it becomes apparent that the time series data is very important and the two job profiles are not enough to define similarity. However, even with a good feature set that works on our test system, it is unlikely that the strategy and trained model will be portable to other systems. Therefore, we conclude that this adapted version of hierarchical clustering based on job profiles isn't suitable to analyze the I/O time series data of jobs and we must compare time series to obtain suitable similarity metrics.



(a) IO-metric



(b) IO-duration

Figure 6.2: The Top 20 largest clusters for two different job-profiles.

Job-IO-Utilization	Job-IO-Problem-Time	Job-IO-Balance	B-coding
4	1	0.438	[118]
4	1	0.445	[368:368:368:368:368:368:374:368:368:368]
4	1	0.458	[496:496]

(a) IO-metrics job profiles

1_md_file_delete	1_md_mod	1_md_other	1_md_read	1_read_bytes	1_read_calls	1_write_bytes	1_write_calls	B-coding
0.009	0.019	0.250	0.028	0.005	0.009	0.005	0.032	[340:510:272]
0.013	0.013	0.245	0.006	0.006	0.006	0.013	0.013	[510:-:-]
0.017	0.017	0.250	0.017	0.000	0.000	0.000	0.017	[14:280]

(b) IO-duration job profiles

Table 6.2: Jobs found in the same cluster with their profiles coding ($\epsilon = 0.03$). Columns containing zeros only are omitted.

6.5 Quantitative Evaluation

This section contains data statistics, clustering progress, and clustering results when applying the five customized algorithms: B_ALL, B_AGGZEROS, Q_LEV, Q_NATIVE, Q_PHASES.

6.5.1 Impact of the User-Defined Similarity

In the introduced algorithms, the user-defined similarity (SIM) that the jobs in a cluster must fulfill to the cluster centroid controls the cluster formation. It is expected that low SIM values produce a smaller number of noisy clusters and a high SIM value produces a large number of clean clusters. We suppose the optimal value is application dependent. Although an optimal SIM value depends on use case and dataset, a parameter exploration may provide important hints to find a good value and achieve optimal cluster qualities.

Figure 6.3 shows the number of clusters created when clustering an increasing total number of jobs for different SIM values; each point represents the number for an analyzed number of jobs in increments of 10,000 jobs. For all algorithms, we can see that with an increase in SIM value, the number of clusters created increases, and the number of total clusters created slows down the more jobs have been processed as jobs are allocated to existing clusters. B_ALL creates most clusters while Q_NATIVE creates the least and Q_PHASES is in between. For SIM of 99%, B* and Q_LEV can barely group jobs together.

Figure 6.4 shows the clustering runtimes for the same experiment. The clustering time depends heavily on the number of created clusters. The reason is that the algorithm tries to put each job in existing clusters first. It iterates over them, and only if it is not able to find a suitable cluster, it creates a new one. The more clusters exist, the longer is the processing time. Since, for low SIM values there is a low number of clusters, the clustering is much faster. Q_PHASES has exceptionally high runtimes due to quadratic runtimes of phase matching. In one case, the runtimes clustering of 10,000 jobs took up to 4.3 hours (the few outliers are not shown in the picture). We suppose this is caused by the quadratic runtimes of the phase matching procedure.

From this initial considerations, it appears that Q_NATIVE is well suited, it generates the least number of clusters and is efficient. However, looking at the overall number of created clusters isn't enough to assess the quantity of the aggregation. Additional quantitative metrics need to be used, such as the number of small clusters.

To understand the aggregation behavior better, alternative visualizations are investigated. In Figure 6.5, the number of clusters created for a given similarity value is plotted. The red line approximates the overall number of clusters, the green line shows how many contain at least two jobs and the blue line shows how many of them contain at least 10 jobs. The maximum number of clusters is equivalent to the number of jobs; it is visualized by the gray line. Coding with 100% similarity are of the same job phenotype, i.e, they have exactly the same length and I/O behavior.

The gradient of the curve shows the generalization capabilities of each algorithm for different SIM values. Apparently, for all algorithms (except Q_LEV and Q_NATIVE) there is a SIM value, where the number of clusters with more than ten and two jobs is decreasing, i.e., the clustering algorithms start to split clusters into individual job clusters. That is something we usually want to prevent, because the algorithms stop finding similar jobs, but focus on refining clusters.

In Figure 6.6, the distribution of the relative cluster size for all jobs. The B* algorithms tend to create many small clusters. Q_PHASES creates a variety of different cluster sizes and Q_NATIVE the biggest clusters. Interestingly after scaling both axes with log10, we can observe a linear behavior for all algorithms.

When looking for an optimal SIM, we consider the following criteria: we need to select a SIM value where a further increase of it doesn't increase the number of clusters significantly, i.e., we can see a flattening curve with increasing numbers of jobs, which indicates that more jobs are placed in clusters and less clusters are created. Q_PHASES and B* algorithms work best for SIM values between 0.7 and 0.9, and for other Q-algorithms the SIM value between 0.90 and 0.99.

Another strategy could be to start with a high SIM value and then decrease it until the clusters are big enough while appearing to be sufficiently similar for the current analysis.

6.6 Cluster Relevance

To assess the quality of the quantitative clustering better and to aid support staff to identify relevant clusters, we define a relevance of clusters as listed in Equation (8). The idea behind the relevance definition is the following: the larger the cluster (in terms of jobs) and the longer the jobs, the more potential load these jobs can produce. Therefore, it is worth investigating these clusters first. Note that the relevance could include the number of occupied nodes as well to effectively indicate the usage of all jobs on the supercomputer. We use the listed

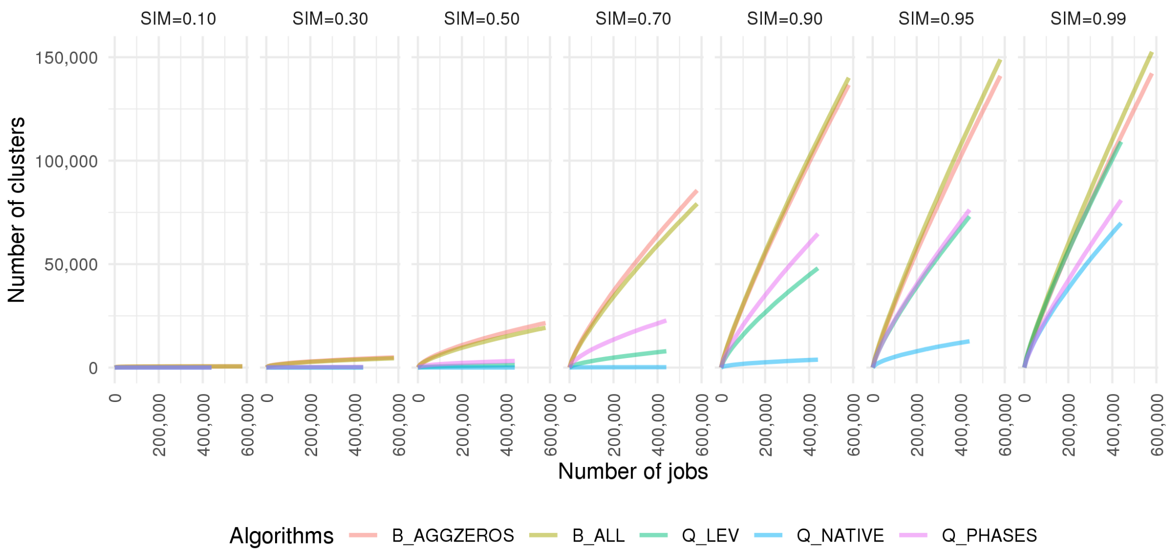


Figure 6.3: Clustering progress.

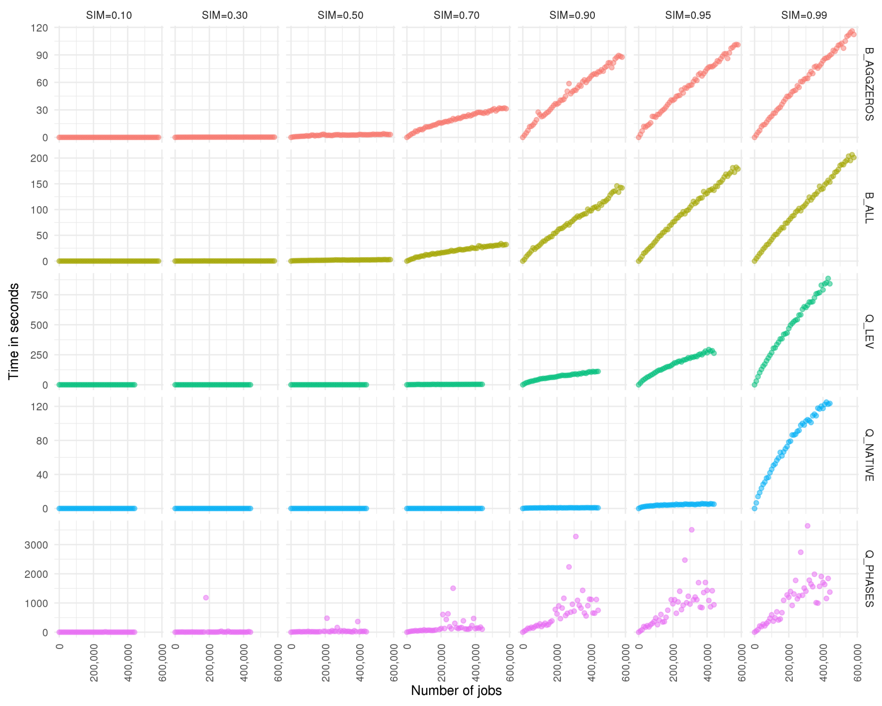


Figure 6.4: Runtime for executing the clustering.

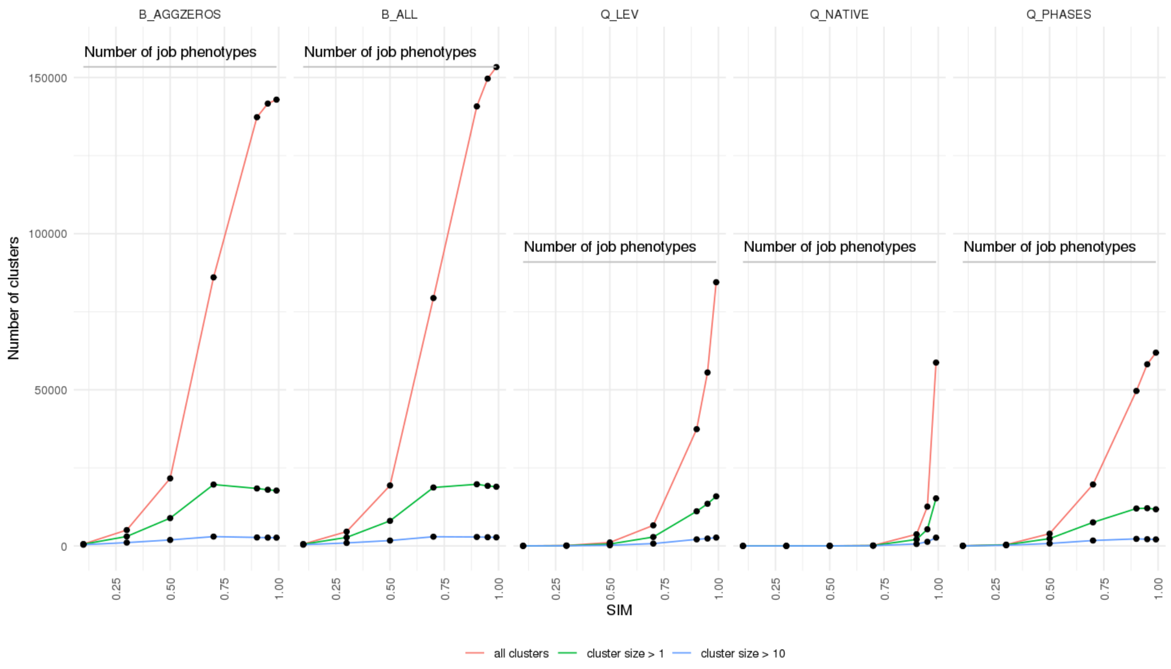


Figure 6.5: Similarity value exploration.

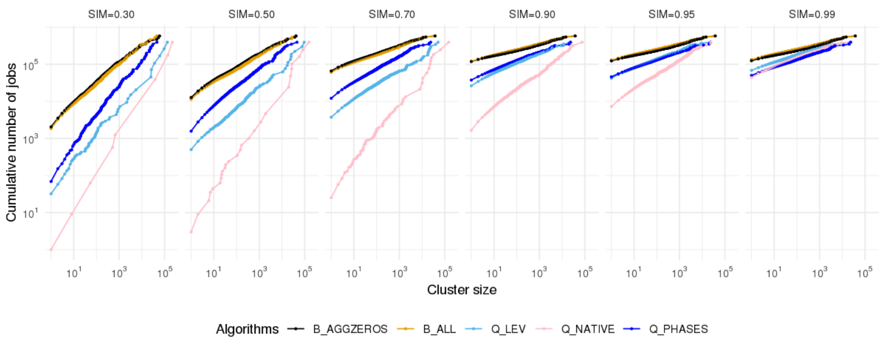


Figure 6.6: Cumulative number of jobs with different sizes.

definition of weighted relevance as an example how it changes rankings.

$$\text{Relevance} = \text{ClusterSize} \cdot \text{MeanJobLength} \tag{8}$$

To give you an impression of cluster qualities for different SIM values, we visualize the Top 10 relevant clusters in Figure 6.7. The figure confirms that each algorithm creates different clusters as the length of the most relevant clusters and amount is dissimilar. For a small SIM value, the first few clusters contain a large quantity of shorter (but not so similar) jobs. However, due to our definition of the relevance metrics, we can observe that smaller clusters with longer jobs are more relevant. This can be as extreme as a cluster with average length of > 40 segments is ranked next to one with average length of 1 segment. We believe that clustering by relevance (potentially extended by the occupied node number) is useful for support staff to identify optimization potential.

6.7 Qualitative Evaluation

The analysis so far does not mean that the quality of the clusters is acceptable. After manual exploration of selected clusters, we noticed some show a SIM-value independent characteristics. To explain them, for each algorithm we investigate one cluster of these in detail.

For the presentation of the clusters, we provide a description of particular observations followed by a table containing statistics, then the codings for centroid and most frequent unique jobs in the cluster – we call such a unique representation a *job phenotype*. At the end, we include an illustration which describes the segment length distribution of jobs in the cluster that indicates how much the algorithm groups across different job runtimes.

6.7.1 Algorithm characteristics

The SIM value selection strategy can vary from use case to use case. As criteria, we choose a SIM value that creates a moderate number of clusters (around 50% of job phenotypes) and keeps its generalization capabilities (the number of clusters with more than 1 job is considerable). For the B-algorithms and Q_PHASES we chose a SIM $= 0.7$, and the other Q-algorithms SIM ≥ 0.9.

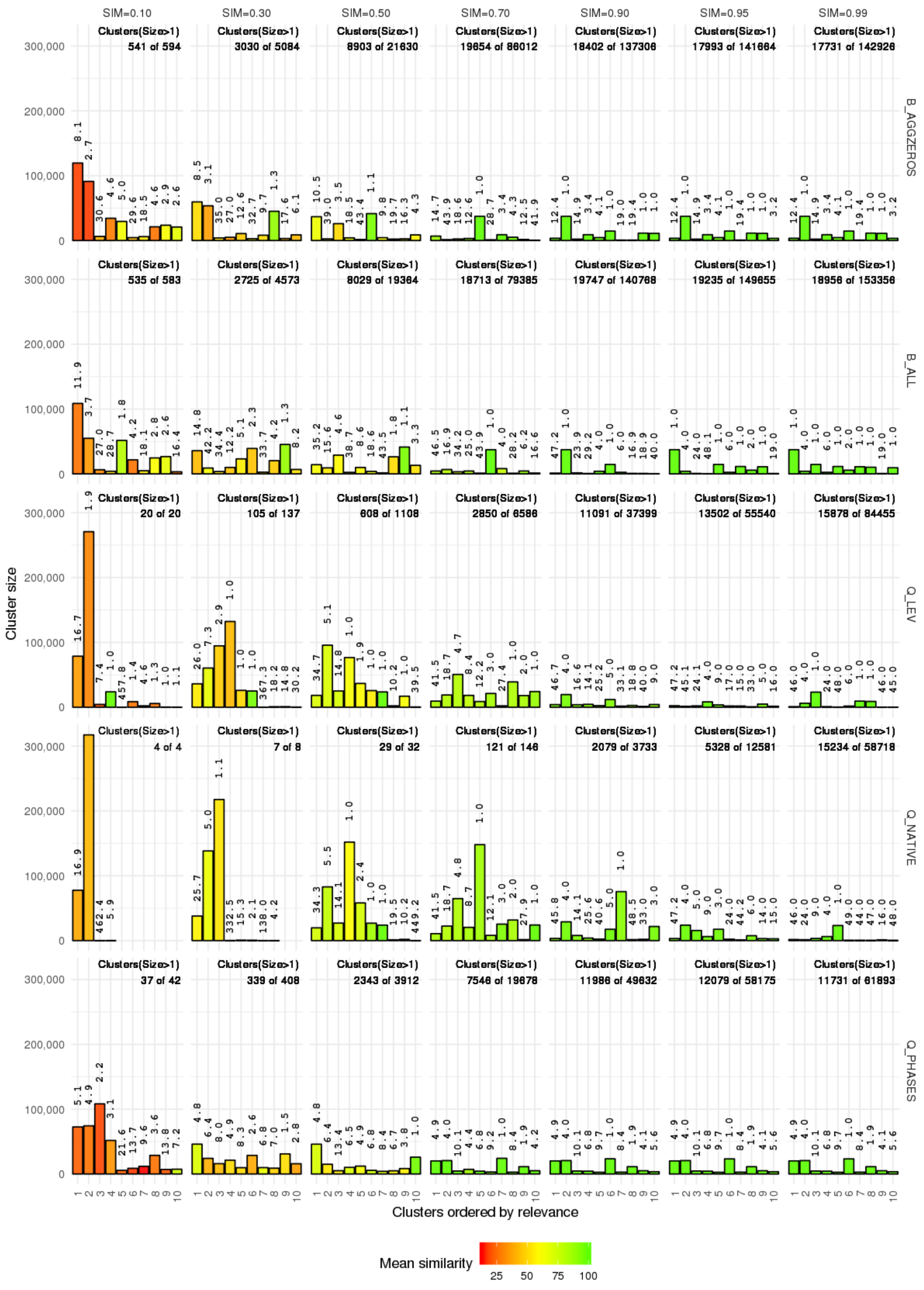


Figure 6.7: Top 10 relevant jobs ordered by relevance. The colors indicate the mean similarity of the jobs in a cluster to the cluster centroid. The number above a bar denotes the mean job length of the cluster.

Coding sequence	Cluster size
[511]	37,272
[32]	14,536
[272]	11,338
[160]	11,014
[128]	10,228
[8]	9,446

Table 6.3: B-algorithms: Top 6 largest clusters created with SIM = 0.7.

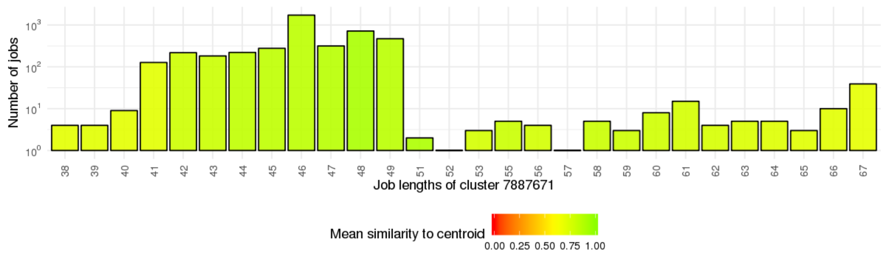
Number of jobs	4378
Number of job phenotypes	3332

(a) Cluster statistics.

B-coding of the centroid	Type
-:-:-:-:-:294:-:-:-:-:32:-:-:-:-:-:-:-:-:-:-:-:32:-:-:-:-:-:-:-:-:-:-:-:32:-:-:-:-:-:-:-:-:-:-	centroid

B-coding of jobs in the cluster	Count
-:-:-:-:-:359:96:-	95
-:-:-:-:-:295:-	62
-:-:-:-:-:-:-:-:-:-:-:4:-	47
-:-:-:-:-:359:-	44
-:-:-:-:6:6:-	40

(b) Centroid and Top 5 job phenotypes.



(c) Length distribution in the cluster.

Figure 6.8: B-algorithm: Information of the selected cluster (SIM=0.7).

B_ALL For the B-algorithms, we show the statistics of the largest clusters as well in Table 6.3 as there is an important aspect. Looking at Table 6.3, the six largest clusters contain the shortest possible jobs (one segment long). In each cluster, all jobs have the identical coding; these short running jobs are likely pre/post-processing jobs.

Therefore, the selected cluster for inspection is further down in the Top 5, information is shown in Figure 6.8. There are about 3300 different types of unique jobs, it contains jobs with a length between 38 and 67. Most jobs in the cluster are shorter than 49 segments, because the main Mistral partitions can allocate jobs for at most 8 hours, and that is the majority of jobs. Since each segment is ten minutes long, the runtime of jobs with 49 segments is about 8.16 hours. The other jobs must be special allocations.

By inspecting individual jobs in Figure 6.8b, we can see that the centroid has 4 segments $\neq 0$ and the jobs are mostly empty. As the similarity of 70% considers empty segments as well, jobs with few I/O segments are found in this cluster.

The following general characteristic observations are made:

1. Job phenotypes can significantly differ from centroid and from each other.
2. Lengths of job phenotypes in a cluster are relatively close to the centroid.

B_AGGZEROS The first clusters are identical to B_ALL, e.g., as shown in Table 6.3. The reason that both algorithms create the same clusters is that the zero aggregation has no effect

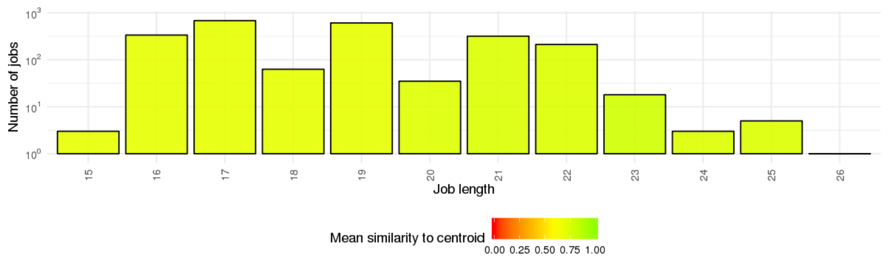
Number of jobs	2285
Number of job phenotypes	452

(a) Cluster statistics.

B-coding of the centroid	Type
-:-:191:272:272:287:287:272:272:272:272:272:272:272:272:272:272:272:272:272:272	centroid

B-coding of jobs in the cluster	Count
272:272:272:272:272:278:286:272:272:272:272:272:272:272:272:272:272	543
272:272:272:272:272:278:286:272:272:272:272:272:272:272:272:272:272:272:272	530
272:272:272:272:272:406:286:272:272:272:272:272:272:272:272:272	96
272:272:272:272:272:272:272:272:272:272:272:278:286:272:272:272	90
272:272:272:272:272:272:272:272:272:272:272:406:286:272:272:272	70

(b) Centroid and Top 5 job phenotypes



(c) Job length distribution in the cluster.

Figure 6.9: B_AGGZEROS algorithm: Information of the selected cluster (SIM=0.7).

on formation of short jobs: the minimum job length required for effective zero aggregation is at least three segments.

We chose again a cluster of the Top 20, information related to the cluster is shown in Figure 6.9. The following is characteristic for this algorithm:

1. Job types can significantly differ from centroid and from each other.
2. I/O intensive clusters appear to be cleaner than B_ALL.
3. If codings contain many zero segments (not included in the example), lengths of job phenotypes in a cluster may be relatively far from the centroid and compared to B_ALL.

Q_LEV Due to the longer Q-codings, smaller changes in the SIM value don't change the clusters as quickly, compared to B-codings. Therefore, for this exploration, we chose a SIM value higher than for B-algorithms. We observe the following cluster characteristics:

1. Similar job lengths.
2. Mostly clean clusters, but a cluster can contain outlier jobs.

We observe that even with a high SIM value, some jobs have a different I/O behavior than the centroid and the rest of the jobs. The reason is that for the Levenshtein distance the distance between value 1 and 2 is the same as for 1 and 8; however, in the former case the performance is more similar than in the latter case.

Information related to the selected cluster is provided in Figure 6.10. In the selected cluster, there is mostly one segment of I/O activity.

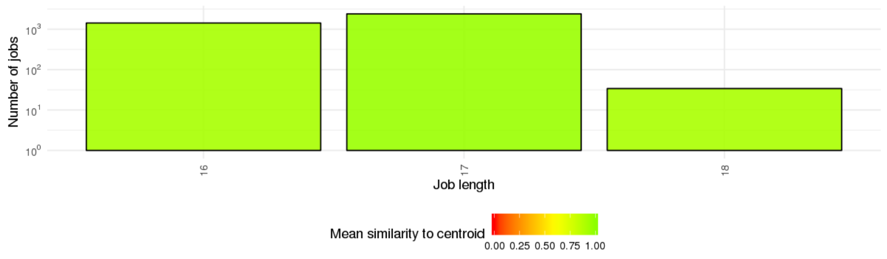
Number of jobs	4453
Number of job phenotypes	1955

(a) Cluster statistics.

Q-coding md_file_delete	md_mod	md_other	Type
-:-:-:-:-:-:-:-:-:-:-:-:2:-:-:-:-	-:-:-:-:-:-:-:-:-:-:-:2:2:-:-:-:-	-:...:-	centroid

Q-coding md_file_delete	md_mod	md_other	Count
-:...:-	-:-:-:-:-:-:-:-:-:-:-:2:-:-:-:-	-:...:-	299
-:...:-	-:-:-:-:-:-:2:-:-:-:-:-:-:-:-:-:-	-:...:-	212
-:...:-	-:-:-:-:-:-:4:-:-:-:-:-:-:-:-:-:-	-:...:-	179
-:...:-	-:...:-	-:-:-:-:-:-:-:-:-:-:-:2:-:-:-	120
-:...:-	-:...:-	-:-:-:-:-:-:-:-:-:-:-:2:-:-:-:-	87

(b) Centroid and Top 5 job phenotypes. The metrics that have no I/O activity are not included in the table.



(c) Length distribution in the cluster.

Figure 6.10: Q_LEV algorithm: Information of the selected cluster (SIM=0.9).

Q_NATIVE The SIM value exploration shows that this algorithm works best with high SIM ≥ 0.9 values. Clustering with SIM=0.99 results in clusters that have equal job lengths. From the definition of the equation, for a similarity of 99% the centroid must be longer than 100 segments to attract jobs with 99 or 101 length. Due to space restrictions, we could not visualize a representative example. Instead of that we take a short one, although it doesn't include all the typical characteristics.

General cluster characteristics:

1. Similar job lengths (even for lower SIM).
2. Jobs are relatively close to the centroid.
3. Low number of outliers.

Information related to the selected cluster is given in Figure 6.11. In the selected cluster, jobs with a length of 4 have mostly the activity of one segment in common.

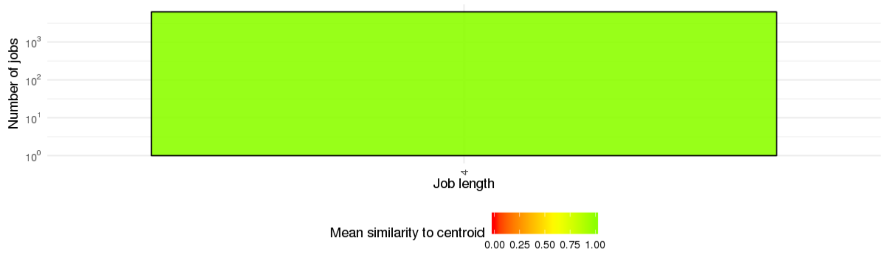
Number of jobs	6246
Number of job phenotypes	23

(a) Cluster statistics.

Q-coding				
md_file_delete	md_mod	md_other	read_bytes	Type
-:...:-	-:...:-	-:-:1:-	-:...:-	centroid

md_file_delete	md_mod	md_other	read_bytes	Count
-:...:-	-:...:-	-:-:1:-	-:...:-	6115
-:...:-	-:-:1:-	-:-:1:-	-:...:-	31
-:...:-	-:...:-	-:-:1:-	-:-:1:-	25
-:1:-:-	-:...:-	-:-:1:-	-:...:-	12
-:...:-	-:1:-:-	-:-:1:-	-:...:-	10

(b) Centroid and Top 5 job phenotypes.



(c) Length distribution in the cluster.

Figure 6.11: Q_NATIVE algorithm: Information of the selected cluster (SIM=0.99).

Q_PHASES Information related to the selected cluster is shown in Figure 6.12. The first thing to mention are the high generalization capabilities of the algorithm, i.e., that many jobs are mapped to a relatively low number of types. The next typical property is shown in Figure 6.12c, where we can see that almost the full range of job lengths is represented in the cluster. This happens, because the Q_PHASES ignores zero segments. For example, for Q_PHASES, the job phenotypes in Figure 6.12b in the row one and two are 100% similar, despite of different lengths. The centroid and other jobs contain very quite similar I/O patterns.

General cluster characteristics:

1. Low number of job phenotypes represented in a cluster.
2. Relatively large number of job lengths represented in a cluster.
3. I/O pattern of jobs in a cluster are similar.

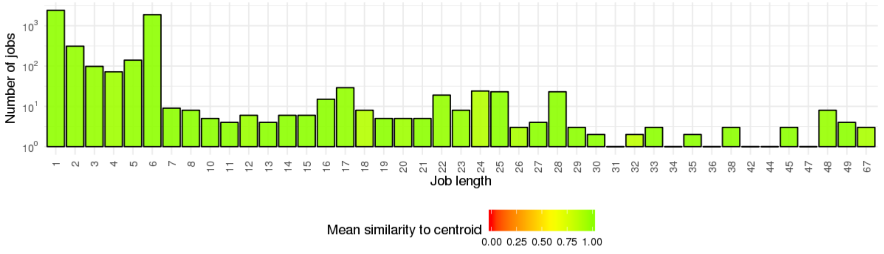
Number of jobs	3015
Number of job phenotypes	254

(a) Cluster statistics.

Q-coding md_file_delete	md_mod	Type
2:-:-:-:-:-	2:-:-:-:-:-	centroid

md_file_delete	md_mod	Count
2	2	958
-:-:-:-:-:-:2:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-	-:-:-:-:-:2:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-	322
2:-:-:-:-:-	2:-:-:-:-:-	309
-:-:2:-	-:-:2:-	140
2:-	2:-	95

(b) Centroid and Top 5 job phenotypes.



(c) Length distribution in the cluster.

Figure 6.12: Q_PHASES algorithm: Information of the selected cluster (SIM=0.7).

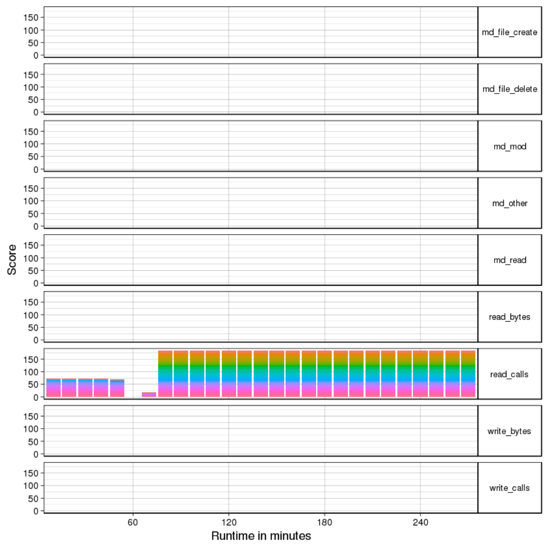


Figure 6.13: One I/O-intense job running on 46 nodes. Score is the sum of all nodes stacked by the node. A color represents one of the nodes.

6.8 Use Case: Tracking an I/O-Intensive Job

The demonstration in this section shows how this approach can be used to identify a cluster of I/O-intensive jobs similar to an existing job.

Firstly, we find an I/O intensive job that we use to identify similar jobs. The selected job is visualized in Figure 6.13. This relatively long job (27 segments) reveals one heavily used metric. The job reads data over its runtime but doesn't have any noteworthy activities in any other metrics. At beginning, only a subset of the nodes is reading most of the data, later more nodes participate in the reading. The amount of transmitted data is not large but the

Number of jobs	8
Number of job phenotypes	8

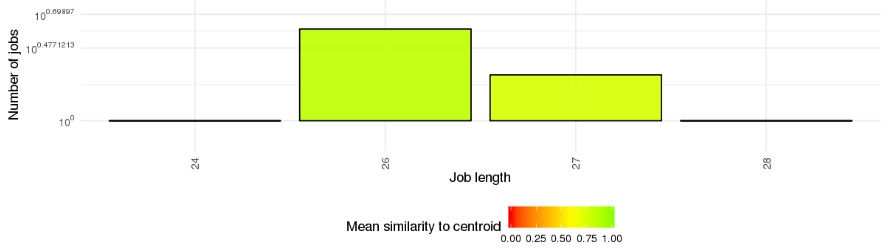
(a) table

Cluster statistics.

B-coding	Type
192:192:192:192:192:192:196:192:192:192:192:192:192:192:192:192:192:192:192:192:192:64:64:64:64:64	job
511:238:192:510:192:224:228:192:192:192:192:192:192:192:192:192:192:192:192:192:64:64:64:64:64	centroid

B-coding	Count
-:224:192:192:192:192:228:192:192:192:192:192:192:192:192:192:192:192:192:192:192:64:64:64:64:64	1
192:192:192:192:192:192:196:192:192:192:192:192:192:192:192:192:192:192:192:192:192:64:64:64:64:64	1
192:192:193:196:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:64:64:64:64	1
192:193:193:198:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:64:64:64:64:64:64	1
207:463:225:495:246:224:198:192:192:192:192:192:192:192:192:192:192:192:192:192:192:64:64:64:64:64:64	1

(b) Job, centroid and Top 5 job phenotypes.



(c) Length distribution in the cluster.

Figure 6.14: B_ALL algorithm: Information of the selected cluster (SIM=0.7).

number of read calls is high and may potentially degrade the file system performance. Now, we identify and investigate the cluster that contains this job for the different algorithms and discuss if these jobs are similar.

Overall, we find this job leads to good conditions for all algorithms, i.e., all the algorithms work well in this use case.

B_ALL Information related to the cluster is in Figure 6.14. Compared to B_AGGZEROS, the cluster is smaller but the jobs have a more similar ending.

B_AGGZEROS Information related to the cluster of the job is in Figure 6.15. We find that the identified jobs in the cluster are sufficiently related.

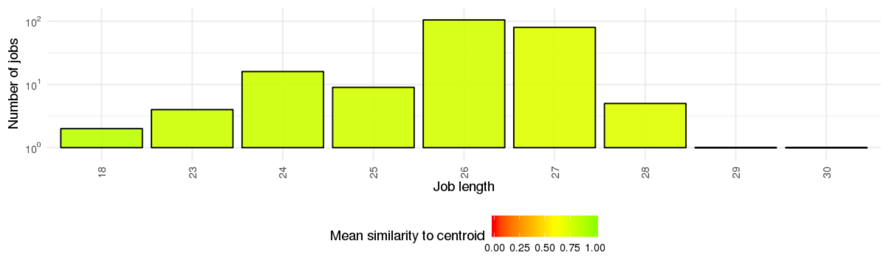
Number of jobs	223
Number of job phenotypes	190

(a) Cluster statistics.

B-coding	Type
192:192:192:192:192:192:196:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:64:64:64:64:64	job
-:-:-:-:-:-:228:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192	centroid

B-coding	Type
192:192:192:192:192:454:230:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192	10
192:192:192:192:192:454:230:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192	10
192:192:192:192:192:454:198:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192	5
-:-:-:-:-:-:228:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192:192	4
192:192:192:192:192:192:192:192:192:192:192:454:230:192:192:192:192:192:192:192:192:192:192:192	3

(b) Job, centroid and Top 5 job phenotypes.



(c) Length distribution in the cluster.

Figure 6.15: B_AGGZEROS algorithm: Information of the selected cluster (SIM=0.7).

Q_LEV Information related to the cluster is in Figure 6.16. Overall, the job is matched partially, jobs in the cluster may contain at the beginning an idle phase.

Notice that there is a discrepancy between the job visualization in Figure 6.13 and the quantum codings. While in Figure 6.16 we can see a less intensive phase in the beginning and a high intensive phase afterwards, the quantum coding contains a more or less single high intensive I/O phase. The reason is that we use different reduction functions. While in the illustration we aggregate segments by the sum() function, for the coding we use mean() function for the same set of values. While sum() is better for visualization, the mean value allows the algorithms to assess data independent of the number of nodes used in a job.

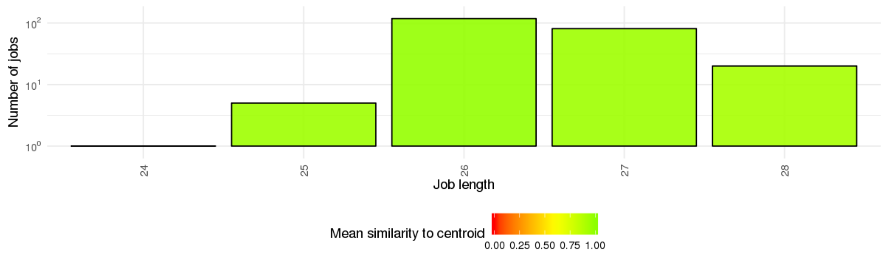
Number of jobs	225
Number of job phenotypes	205

(a) Cluster statistics.

Q-coding		
md_other	read_calls	Type
-:...:-	3:3:8:8:7:4:5:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	job
-:...:-	-:-:-:-:-:-:6:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	centroid

md_other	read_calls	Count
-:...:-	-:-:-:-:-:-:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	5
-:...:-	-:-:-:-:-:-:2:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	3
-:...:-	8:8:8:8:8:2:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	3
-:...:-	8:8:8:8:8:5:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	3
-:-:-:3:-	-:-:-:-:-:-:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	3

(b) Job, centroid and Top 5 job phenotypes.



(c) Length distribution in the cluster.

Figure 6.16: Q_LEV algorithm: Information of the selected cluster (SIM=0.9).

Q_NATIVE Information related to the cluster is in Figure 6.17. Again, the jobs look similar, however, for read calls the increase in performance and the drop in Segment 5 is not captured.

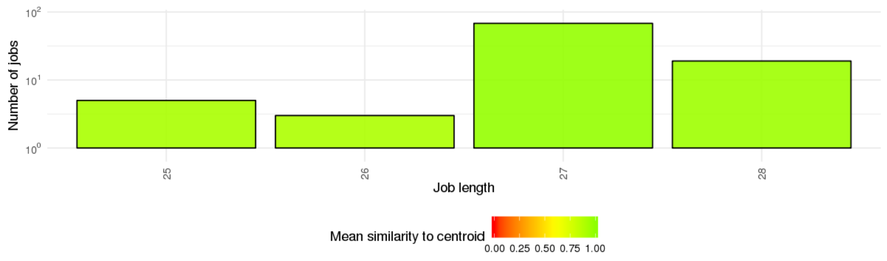
Number of jobs	95
Number of job phenotypes	94

(a) Cluster statistics.

Q-coding read_calls	Type
3:3:8:8:7:4:5:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	job
-:-:-:-:-:-:5:8:8:8:8:8:8:8:-:8:8:8:8:8:8:8:8:8:8:8:8	centroid

read_calls	Count
-:-:-:-:-:-:3:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	2
-:-:-:-:-:-:2:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:7:8:8:8:8	1
-:-:-:-:-:-:4:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	1
-:-:-:-:-:-:5:8:8:8:8:8:8:8:-:8:8:8:8:8:8:8:8:8:8:8:8	1
-:-:-:-:-:-:5:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	1

(b) Job and centroid coding sequences.



(c) Length distribution in the cluster.

Figure 6.17: Q_NATIVE algorithm: Information of the selected cluster (SIM=0.90).

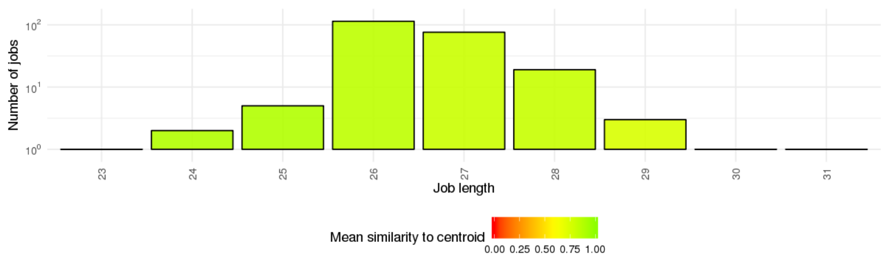
Number of jobs	222
Number of job phenotypes	202

(a) Cluster statistics.

Q-coding md_other	read_calls	Type
-:...:-	3:3:8:8:7:4:5:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	job
-:...:-	-:-:-:-:-:-:6:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	centroid

md_other	read_calls	Count
-:...:-	-:-:-:-:-:-:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	5
-:...:-	-:-:-:-:-:-:2:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	3
-:...:-	8:8:8:8:8:2:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	3
-:...:-	8:8:8:8:8:5:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	3
-:-:-:3:-	-:-:-:-:-:-:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8:8	3

(b) Job, centroid and Top 5 job phenotypes.



(c) Length distribution in the cluster.

Figure 6.18: Q_PHASES algorithm: Information of the selected cluster (SIM=0.7).

Q_PHASES Information related to the cluster is in Figure 6.18. Even with a smaller SIM value, the jobs appear related while they cover a variety of job length as well.

6.9 Discussion

We evaluated similarity between jobs quantitatively and qualitatively for the different algorithms. As most jobs contain only a subset of I/O-intensive segments, the assessment of similarity should ignore these phases. There may be specific use cases where the exact job-length play a role but generally it appears that a user or support member should be interested in similar I/O-patterns as otherwise the clusters will be polluted with irrelevant jobs.

Coding. The coding of the time series data has a big impact on the potential similarity that can be uncovered. We explored binary, quantized values.

We noticed in the experiments with native coding that the amount of generated clusters skyrocketed. The reason is that similarity for short jobs exceed a high SIM value quickly and this leads to the creation of new clusters, which in turn leads to long clustering runtimes. An example illustrates a typical case: Assume there are two jobs: one running on 16 nodes and other running on 32 nodes. Both are one segment long and only one I/O node that writes data to storage with a moderate performance. Without quantization, we would represent these jobs by the following sequences: [..., [0.0625], . . .] and [..., [0.03125], ...], where only active metrics have values larger than zero. Using the similarity function without quantization, i.e., with mean performance, would result in a similarity of 50%, and for SIM > 0.5 they would be placed in separate clusters.

The Q-coding quantizes the data, making them identical, and filters many jobs automatically. Segments with mean performance less than 0.125 are rounded to zero and zero sequences are removed from the dataset.

Rounding is not necessary for Q_PHASES. It can work directly with floating-point numbers. As it generalized better for smaller similarity, it wouldn't create that many clusters as for the other methods. The initial motivation behind the use of quantum coding was to apply string methods such as Levenshtein and make data comparable with other algorithms.

Lengths-invariance. With the Levenshtein distance, we attempted to resolve the issue of lengths-invariance allowing the assignment of jobs of different length into the same cluster. However, the Levenshtein distance cannot consider the performance of a segment. For example, the following three codings would be considered to be all different by one symbol, since they differ in one position only.

```
phase_coding_1 : [2:2:2:2:2:9:2:2]
phase_coding_2 : [2:2:2:2:2:8:2:2]
phase_coding_3 : [2:2:2:2:2:-:2:2]
```

Intuitively, we would say that phase_coding_1 and phase_coding_2 are more similar than phase_coding_2 and phase_coding_3, because the difference at 6th position is smaller for 8 and 9 than to 8 to 0. Short runtimes lead to more polluted clusters.

Here is another example that illustrated the issue of noise in clusters for Levenshtein: Suppose two jobs [-:6:-:-] and [-:388:174:-] are in the same cluster with the centroid [-:388:-:-]. After applying one replacement operation (replace 6 by 388) on the first job [-:6:-:-] we can obtain the centroid [0:388:0:0]. Similarly, we can replace one element in the second job (replace 174 by -) to obtain the same centroid According to the algorithm we obtain a similarity of 75% in both cases, but intuitively we would say that these jobs are completely different, not even close to the 75% mark. We can also easily construct another sequence, e.g., [-:389:-:-], where a similarity of 75% is justified. For this reason, we found that the Levenshtein distance is generally not suitable.

I/O Phases. The current version of the Q_PHASES algorithm ignores NonIO parts (sequences of zeros) of the monitoring data, and handles segment sequences like [-,-,-,-,1] and [1] equally:

```
job1_metric1 : [-,-,-,-,1]
job1_metric1 : [1]
```

These sequences would produce different average I/O loads on the storage. Depending on the use case, we would judge these jobs exhibit different I/O behaviour as the ratio between computational and I/O load is different.

One could argue that the phase definition (of separating phases by idle phases) isn't capturing the behavior sufficiently. Currently, phases are defined for each metric individually, without consideration of the overall behavior on other metrics. For illustration, consider the following two jobs:

```
job1_metric1 : [-:1]
job1_metric2 : [1:-]
job2_metric1 : [1:-]
job2_metric2 : [1:-]
```

Currently, the algorithm recognizes the I/O patterns to be 100% similar as it extracts in all cases a phase of [1]. Alternatively, one could say that running two metrics at the same time is another I/O pattern, as running them shifted, because on a storage system the jobs would produce different I/O loads. For jobs with many I/O phases, this situation is less likely but for shorter jobs, it may happen. The benefit of ignoring these aspects is simplicity; this clustering reduces the number of clusters and results are easier to understand.

Characterization of I/O. The main questions for a user that applies a clustering algorithm remain: how to define similarity. Depending on the use case, the relevant I/O-behavior and characteristics involved in job comparison must be defined to allow identification of jobs depending on temporal behavior. In our case, we discussed similarity based on profiles and time series data. We explored various variants for the handling of fluctuation in the time series data that consider time series of different length, omit empty phases, and allow some reordering of phases. To answer these questions, a discussion with the community about the use cases and a study of more use cases is necessary. It appears likely that multiple similarity metrics must be defined that serve different purposes.

7 Conclusion

In this work, we described various approaches to cluster 1 million jobs based on their I/O behavior. Clustering of jobs is important for data-center support staff to focus their effort on relevant jobs. Therefore, we pre-processed the periodically gathered node metrics by converting this fine-grained resolution into 10-minute segments using various stacks. Our contribution is the systematic evaluation and discussion of various approaches for the clustering. The success of clustering algorithms depends on the right feature selection and data transformation.

After a series of experiments with general purpose algorithms on profile data, we could not achieve acceptable results for the clustering of originally time series data. While we could have applied traditional approaches such as k-means on profiles, we found that the large number of classes and unknown knowledge about the jobs doesn't suit this use case. We believe this is due to the nature of reducing the time series to flat job profiles, which isn't suitable to capture the behavior of the IO jobs. Even if a better solution would be found, there is no guarantee that the generated model could be applied on another system.

In the evaluation, we investigated the quantitative and qualitative behavior of different algorithms. The investigation of clusters shows that particularly clusters with little I/O activity are noisy and do not meet our expectation. Depending on the similarity chosen by the user, there are many different clusters with phenotypes of jobs found. Thus, exceeding the ability for manual inspection and manual labeling.

Our analysis shows that considering the relative I/O performance and I/O phases improves the clustering. The integration of the awareness of I/O performance produces better, but still not perfect results. Therefore, we found that the Levenshtein distance is generally not suitable for similarity of I/O behavior of jobs.

Absolute coding aggregates three dimensions (Metric, Nodes, and FileSystems) resulting in a nine times shorter coding sequence than quantum coding, which aggregates only two dimensions (Nodes, and FileSystems). Using quantum coding allows a more precise job comparison than binary coding because it contains information for all metrics. However, using the actual native floating-point value is problematic for the developed algorithms as for high similarity it will generate too many clusters. Therefore, we consider the quantization is appropriate for this use case.

The phase matching algorithm detects phases and differentiates performance values. In our analysis, even small SIM values produced good results. We found that this algorithm identifies similar jobs sufficiently good to be usable on production systems. The cluster relevance computation makes the algorithm even more valuable, since it aids data center employees to select jobs of relevance.

We believe that the community must identify and define suitable similarity metrics for the different analysis purposes.

7.1 Future Work

To address the issue with the large number of clusters we will research three additional functions: 1) filter irrelevant clusters, 2) sort clusters by criteria, and 3) automatic labeling.

We intend to conduct a survey and discussion with users to answer the question what similarity of I/O jobs means. We will extend relevance to cover node count as well to represent the actual costs of running a job better.

We see a high potential in the new clustering algorithms to apply them on the fly to jobs of interest. For this case, the jobs could be grouped based on distance to this particular job and allowing the user to modify the similarity online.

Acknowledgment

We thank the reviewers Jean-Thomas Acquaviva, Suren Byna, Adrian Jackson, and George Markomanolis for their constructive feedback.

References

[AGG+14] Dong Ahn, Jim Garlick, Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz. Flux: A Next-Generation Resource Management Framework for Large HPC Centers. pages 9–17, 09 2014.

[BK18] Eugen Betke and Julian Kunkel. Benefit of DDN's IME-FUSE for I/O Intensive HPC Applications. In Rio Yokota, Michèle Weiland, John Shalf, and Sadaf Alam, editors, *High Performance Computing*, pages 131–144, Cham, 2018. Springer International Publishing.

[BK20] Eugen Betke and Julian Kunkel. The Importance of Temporal Behavior when Classifying Job IO Patterns Using Machine Learning Techniques. Lecture Notes in Computer Science. Springer, 2020.

[Car15] Philip Carns. Darshan. In *High performance parallel I/O*, Computational Science Series, pages 309–315. Chapman & Hall/CRC, 2015.

[CHA$^+$11] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)*, 7(3):8, 2011.

[EB20] Julian Kunkel Eugen Betke. Semi-automatic Assessment of I/O Behavior by Inspecting the Individual Client-Node Timelines — An Explorative Study on 10^6 Jobs. In *2014 43rd International Conference on Parallel Processing Workshops*. ISC Events, 2020.

[EMAM16] Farsarakis E., Weiland M., Jackson A., and Parson M. Monitoring and evaluating I/O performance of HPC systems, 2016.

[GW99] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, Berlin/Heidelberg, 1999.

[Kar19] Karthee Sivalingam and Harvey Richardson and Adrian Tate and Martin Lafferty. Lassi: Metric based i/o analytics for hpc, 2019.

[KB19] Julian Kunkel and Eugen Betke. Tracking User-Perceived I/O Slowdown via Probing. In *High Performance Computing: ISC High Performance 2019 International Workshops, Frankfurt/Main, Germany, June 20, 2019, Revised Selected Papers*, Lecture Notes in Computer Science. Springer, 07 2019.

[KBB$^+$19] Julian Kunkel, Eugen Betke, Matt Bryson, Philip Carns, Rosemary Francis, Wolfgang Frings, Roland Laifer, and Sandra Mendez. Tools for Analyzing Parallel I/O. In Rio Yokota, Michele Weiland, John Shalf, and Sadaf Alam, editors, *High Performance Computing: ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected Papers*, number 11203 in Lecture Notes in Computer Science, pages 49–70. ISC Team, Springer, 01 2019.

[KM18] Julian Martin Kunkel and George S. Markomanolis. Understanding Metadata Latency with MDWorkbench. In Rio Yokota, Michèle Weiland, John Shalf, and Sadaf Alam, editors, *High Performance Computing*, pages 75–88, Cham, 2018. Springer International Publishing.

[KZH$^+$14] Julian Kunkel, Michaela Zimmer, Nathanael Hübbe, Alvaro Aguilera, Holger Mickler, Xuan Wang, Andrij Chut, Thomas Bönisch, Jakob Lüttgau, Roman Michel, and Johann Weging. The SIOX Architecture – Coupling Automatic Monitoring and Optimization of Parallel I/O. In Julian Kunkel, Thomas Ludwig, and Hans Meuer, editors, *Supercomputing*, Supercomputing, pages 245–260. ISC events, Lecture Notes in Computer Science, 2014.

[LCLA19] Weihao Liang, Yong Chen, Jialin Liu, and Hong An. CARS: A contention-aware scheduler for efficient resource management of HPC storage systems. *Parallel Computing*, 87:25 – 34, 2019.

[LWS+18] Glenn K Lockwood, Nicholas J Wright, Shane Snyder, Philip Carns, George Brown, and Kevin Harms. TOKIO on ClusterStor: Connecting standard tools to enable holistic I/O performance analysis. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2018.

[NAW+96] Wolfgang E Nagel, Alfred Arnold, Michael Weber, Hans-Christian Hoppe, and Karl Solchenbach. Vampir: Visualization and analysis of mpi resources. 1996.

[NCHD19] Ngan Nguyen, Yong Chen, Jon Hass, and Tommy Dang. HiperJobViz: Visualizing Resource Allocations in HPCC via Multivariate Health-Status Data, 2019.

[SRTL19] Karthee Sivalingam, Harvey Richardson, Adrian Tate, and Martin Lafferty. LASSi: Metric based I/O analytics for HPC. *CoRR*, abs/1906.03884, 2019.

[WBH+16] Matthias Weber, Ronny Brendel, Tobias Hilbrich, Kathryn Mohror, Martin Schulz, and Holger Brunst. Structural clustering: a new approach to support performance analysis at scale. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 484–493. IEEE, 2016.

[WOW+14] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and W. Yu. Burstmem: A high-performance burst buffer system for scientific applications. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 71–79, 2014.

Reviews

This section is optional for reviewers and shows their assessment that lead to the acceptance of the original manuscript. Reviewers may or may not update their review for a major update of the paper, the exact trail is available in GitHub repository of this article. The reviews are part of the article and validate the acceptance. Please check the details on the JHPS webpage.

Reviewer: JT Acquaviva, Date: 2020-10-07

Overall summary and proposal for acceptance The paper address a key topic in I/O analysis. As instrumentation and profiling are starting to get in place in data centers the amount of information generated in unmanageable at the human scale and methods need to be developed to summarize the I/O profile ad provide insight. Taking this problem at its inception the article proposes, discuss and compares several ways to describe an job in respect of its I/O characteristics. Not only the metrics are discussed but also the level of accuracy for the measurements. Thus the authors explore the trade-off between accurate description and the need to accept loss of information to build clusters. The quality of the clustering is analyzed and overall the authors are able to summarize 100.000 jobs in a limited amount of clusters. Some qualitative analysis is performed on the clustering. Overall the paper address an important problem, with broad implications for I/O architectures, job scheduling and work-flow organization in a data center. The paper is well written and clear. The data set is large enough to back-up the relevance of the method. It remains that data are all coming from a single site (DKRZ) which is OK to assess the method but somehow limit the scope of the findings.

Scope Yes, it's a good fit.

Significance Above average

Readability Good

Presentation Good

References Yes

Correctness Yes

Reviewer: Suren Byna, Date: 2020-07-07

Overall summary and proposal for acceptance This paper describes an exploration to cluster one million jobs that ran on the Mistral system at DKRZ. The authors used 10,000 jobs for training and then used the 1 million jobs for clustering. It was found that general purpose clustering algorithms were not creating meaningful clusters, hence, the authors explored quantitative and qualitative analysis of clusters using knowledge of I/O patterns.

What does the "score" mean in Figure 2.2? Please define it as it is not obvious based on the figure's caption.

In general, the motivation of this paper is interesting. One thing I wasn't clear at the end, may be due to my lack of expertise in clustering, which clustering method should a data center use. Understandably, clustering depends heavily on similarity, but selection of a similarity could be a daunting task for users.

Overall, this paper is interesting to the I/O community, as the effort is attempting to find clustering of jobs based on I/O, which is difficult, and defining similarity requires more work. Added minor edits as suggestions in the text.

The paper could be accepted for publication after the authors address minor editing suggestions and adding some minor details.

Scope Yes, it's a good fit.

Significance Above average

Readability Good

Presentation Good

References Yes

Correctness Yes

Reviewer: George Markomanolis, Date: 2020-11-19

Overall summary and proposal for acceptance During the profiling of the I/O on a supercomputer to analyze data, a scientist has to handle a problem, there are a lot of data from various jobs. New methodologies need to be developed for more efficient analysis. The authors propose an approach that is applied on Mistral system at DKRZ to map the I/O on some specific metrics where they can apply clustering techniques to avoid having a million different profiling data to analyze. Moreover, the authors did quantitative and qualitative analysis regarding their approach. They evaluated many algorithms and the coding technique helped them to use the time series data to study the similarity. The outcome was to have a small number of clusters that could analyze. There are many frameworks that produce a lot of data, thus this work is really important and could potentially be also used on other big systems. The paper is lengthy but needs to provide a lot of details for the reader to understand better. It would be great if more sites were included in these experiments but this could be future work.

Scope Yes.

Significance Above average

Readability Good

Presentation Good

References Yes

Correctness Yes

Reviewer: Adrian Jackson, Date: 2021-01-25

Overall summary and proposal for acceptance This paper, outlining as it does a study of I/O job metrics and methods for automatically analysing I/O usage data on large scale systems, is generally of publishable quality, and presents interesting research.

There are some places where the language or the discussion in the paper needs to be addressed, and this has been added in comments in the paper. Of most importance are the figures Figure 6.15 and Figure 6.14 where I think the labels are the wrong way around at the moment. There are also some issues around how levenshtein distance has been used in the research that could be improved or at least clarified, because it is not currently clear how text manipulations are being undertaken, i.e. in the example [-:6:-:-] -¿ [-:388:-:-] is described as requiring one change, but for classical Levenshtein distance that should be a minimum of 3 changes.

There are other minor areas where the text could be improved, but in general the rest of the paper is good.

I would suggest this paper can be accepted provided the corrections outlined in the comments have been sufficiently addressed.

Scope Yes, the investigation of I/O monitoring metrics and analysis is a good fit to the journal

Significance The work has good significance in the topic of I/O metrics and analysis

Readability In general the readability is good, although there are a number of places where the language could be improved or where text is not completely understandable. Comments have been added in the text outlining these issues.

Presentation Good

References Good

Correctness The research is thorough and well documented.