Toward a Workflow for Identifying Jobs with Similar I/O Behavior Utilizing Time Series Analysis

Julian Kunkel[1] and Eugen Betke[2]

[1] Georg-August-Universität Göttingen/GWDG `julian.kunkel@gwdg.de`
[2] ECMWF `eugen.betke@ecmwf.int`

Abstract. One goal of support staff at a data center is to identify inefficient jobs and to improve their efficiency. Therefore, a data center deploys monitoring systems that capture the behavior of the executed jobs. While it is easy to utilize statistics to rank jobs based on the utilization of computing, storage, and network, it is tricky to find patterns in 100,000 jobs, i.e., is there a class of jobs that aren't performing well. Similarly, when support staff investigates a specific job in detail, e.g., because it is inefficient or highly efficient, it is relevant to identify related jobs to such a blueprint. This allows staff to understand the usage of the exhibited behavior better and to assess the optimization potential.

In this article, our goal is to identify jobs similar to an arbitrary reference job. In particular, we sketch a methodology that utilizes temporal I/O similarity to identify jobs related to the reference job. Practically, we apply several previously developed time series algorithms. A study is conducted to explore the effectiveness of the approach by investigating related jobs for a reference job. The data stem from DKRZ's supercomputer Mistral and include more than 500,000 jobs that have been executed for more than 6 months of operation. Our analysis shows that the strategy and algorithms bear the potential to identify similar jobs, but more testing is necessary.

1 Introduction

Supercomputers execute 1000s of jobs every day. Support staff at a data center have two goals. Firstly, they provide a service to users to enable them the convenient execution of their applications. Secondly, they aim to improve the efficiency of all workflows – represented as batch jobs – in order to allow the data center to serve more workloads.

In order to optimize a single job, its behavior and resource utilization must be monitored and then assessed. Rarely, users will liaise with staff and request a performance analysis and optimization explicitly. Therefore, data centers deploy monitoring systems and staff must pro-actively identify candidates for optimization. Monitoring and analysis tools such as TACC Stats [7], Grafana [4], and XDMod [16] provide various statistics and time-series data for job execution.

The final publication is available at Springer through the DOI 10.1007/978-3-030-90539-2_10

The support staff should focus on workloads for which optimization is beneficial, for instance, the analysis of a job that is executed once on 20 nodes may not be a good return of investment. By ranking jobs based on their utilization, it is easy to find a job that exhibits extensive usage of computing, network, and I/O resources. However, would it be beneficial to investigate this workload in detail and potentially optimize it? For instance, a pattern that is observed in many jobs bears potential as the blueprint for optimizing one job may be applied to other jobs as well. This is particularly true when running one application with similar inputs, but also different applications may lead to similar behavior. Knowing details about a problematic or interesting job may be transferred to similar jobs. Therefore, it is useful for support staff (or a user) that investigates a resource-hungry job to identify similar jobs that are executed on the supercomputer.

It is non-trivial to identify jobs with similar behavior from the pool of executed jobs. Re-executing the same job will lead to slightly different behavior, a program may be executed with different inputs or using a different configuration (e.g., number of nodes). Job names are defined by users; while a similar name may hint to be a similar workload, finding other applications with the same I/O behavior would not be possible.

In the paper [2], we developed several distance measures and algorithms for the clustering of jobs based on the time series and their I/O behavior. These distance measures can be applied to jobs with different runtimes and the number of nodes utilized, but differ in the way they define similarity. They showed that the metrics can be used to cluster jobs, however, it remained unclear if the method can be used by data center staff to explore similar jobs effectively. In this paper, we refine these algorithms slightly, include another algorithm, and apply them to rank jobs based on their temporal similarity to a reference job.

We start by introducing related work in Section 2. In Section 3, we describe briefly the data reduction and the algorithms for similarity analysis. Then, we perform our study by applying the methodology to a reference job, therewith, providing an indicator for the effectiveness of the approach to identify similar jobs. In Section 5, the reference job is introduced and quantitative analysis of the job pool is made based on job similarity. In Section 6, the 100 most similar jobs are investigated in more detail, and selected timelines are presented. The paper is concluded in Section 7.

2 Related Work

Related work can be classified into distance measures, analysis of HPC application performance, inter-comparison of jobs in HPC, and I/O-specific tools.

The ranking of similar jobs performed in this article is related to clustering strategies. Levenshtein (Edit) distance is a widely used distance metric indicating the number of edits needed to convert one string to another [14]. The comparison of the time series using various metrics has been extensively investigated. In [9], an empirical comparison of distance measures for the clustering of multivariate time series is performed. 14 similarity measures are applied to 23 data sets. It shows that no similarity measure produces statistically significant better results than another. However, the Swale scoring model [13] produced the most disjoint clusters.

The performance of applications can be analyzed using one of many tracing tools such as Vampir [18] that record the behavior of an application explicitly or implicitly by collecting information about the resource usage with a monitoring system. Monitoring systems that record statistics about hardware usage are widely deployed in data centers to record system utilization by applications. There are various tools for analyzing the I/O behavior of an application [10].

For Vampir, a popular tool for trace file analysis, in [18] the Comparison View is introduced that allows them to manually compare traces of application runs, e.g., to compare optimized with original code. Vampir generally supports the clustering of process timelines of a single job, allowing to focus on relevant code sections and processes when investigating many processes.

In [8], 11 performance metrics including CPU and network are utilized for agglomerative clustering of jobs, showing the general effectiveness of the approach. In [15], a characterization of the NERSC workload is performed based on job scheduler information (profiles). Profiles that include the MPI activities have shown effective to identify the code that is executed [5]. Many approaches for clustering applications operate on profiles for compute, network, and I/O [6, 11, 1]. For example, Evalix [6] monitors system statistics (from proc) in 1-minute intervals but for the analysis, they are converted to a profile removing the time dimension, i.e., compute the average CPU, memory, and I/O over the job runtime.

PAS2P [12] extracts the I/O patterns from application traces and then allows users to manually compare them. In [19], a heuristic classifier is developed that analyzes the I/O read/write throughput time series to extract the periodicity of the jobs – similar to Fourier analysis. The LASSi tool [17] periodically monitors Lustre I/O statistics and computes a "risk" factor to identify I/O patterns that stress the file system. In contrast to existing work, our approach allows a user to identify similar activities based on the temporal I/O behavior recorded by a data center-wide deployed monitoring system.

3 Methodology

The purpose of the methodology is to allow users and support staff to explore all executed jobs on a supercomputer in order of their similarity to the reference job. Therefore, we first need to define how a job's data is represented, then describe the algorithms used to compute the similarity, and, the methodology to investigate jobs.

3.1 Job Data

On the Mistral supercomputer at DKRZ, the monitoring system [3] gathers in ten seconds intervals on all nodes nine I/O metrics for the two Lustre file systems together with general job metadata from the SLURM workload manager. The results are 4D data (time, nodes, metrics, file system) per job. The distance measures should handle jobs of different lengths and node count. In the open-access article [2], we discussed a variety of options from 1D job-profiles to data reductions to compare time series data and the general workflow and pre-processing in detail. We will be using this representation. In a nutshell, for each job executed on Mistral, they partitioned it into 10 minutes segments[3] and compute the arithmetic mean of each metric, categorize the value into NonIO (0), HighIO (1), and CriticalIO (4) for values below 99-percentile, up to 99.9-percentile, and above, respectively. The values are chosen to be 0, 1, and 4 because we arithmetically derive metrics: naturally, the value of 0 will indicate that no I/O issue appears; we weight critical I/O to be 4x as important as high I/O. This strategy ensures that the same approach can be applied to other HPC systems regardless of the actual distribution of these statistics on that data center. After the mean value across nodes is computed for a segment, the resulting numeric value is encoded either using binary (I/O activity on the segment: yes/no) or hexadecimal representation (quantizing the numerical performance value into 0-15) which is then ready for similarity analysis. By pre-filtering jobs with no I/O activity – their sum across all dimensions and time series is equal to zero – the dataset is reduced from 1 million jobs to about 580k jobs.

3.2 Algorithms for Computing Similarity

We reuse the B and Q algorithms developed in [2]: B-all, B-aggz(eros), Q-native, Q-lev, and Q-phases. They differ in the way data similarity is defined; either the time series is encoded in binary or hexadecimal quantization, the distance measure is the Euclidean distance or the Levenshtein distance. B-all determines the similarity between binary codings by means of Levenshtein distance. B-aggz is similar to B-all, but computes similarity on binary codings where subsequent segments of zero activities are replaced by just one zero. Q-lev determines the similarity between quantized codings by using Levenshtein distance. Q-native uses a performance-aware similarity function, i.e., the distance between two jobs for a metric is $\frac{|m_{\text{job1}} - m_{\text{job2}}|}{16}$. One of our basic considerations is that a short job may run longer, e.g, when restarted with a larger input file (it can stretch the length of the I/O and compute phases) or when run with more simulating steps. There are more alternatives how a longer job is related to a shorter job but we do not consider them for now. In this article, we consider these different behavioral patterns and attempt to identify situations where the I/O pattern of a long job is contained in a shorter job. Therefore, for jobs with different lengths, a sliding-windows approach is applied which finds the location for the shorter job in the long job with the highest similarity. Q-phases extracts phase information and performs a phase-aware and performance-aware similarity computation.

[3] We found in preliminary experiments that 10 minutes reduces compute time and noise, i.e., the variation of the statistics when re-running the same job.

The Q-phases algorithm extracts I/O phases from our 10-minute segments and computes the similarity between the most similar I/O phases of both jobs.

3.3 Methodology

Our strategy for localizing similar jobs works as follows:

- A user[4] provides a reference job ID and selects a similarity algorithm.
- The system iterates over all jobs of the job pool, computing the similarity to the reference job using the specified algorithm.
- It sorts the jobs based on the similarity to the reference job.
- It visualizes the cumulative job similarity allowing the user to understand how job similarity is distributed.
- The user starts the inspection by looking at the most similar jobs first.

The user can decide about the criterion when to stop inspecting jobs; based on the similarity, the number of investigated jobs, or the distribution of the job similarity. For the latter, it is interesting to investigate clusters of similar jobs, e.g., if there are many jobs between 80-90% similarity but few between 70-80%.

For the inspection of the jobs, a user may explore the job metadata, search for similarities, and explore the time series of a job's I/O metrics.

4 Reference Job

For this study, we chose the reference job called Job-M: a typical MPI parallel 8-hour compute job on 128 nodes that write time series data after some spin up. The segmented timelines of the job are visualized in Figure 1 – remember that the mean value is computed across all nodes on which the job ran. This coding is also used for the Q algorithms, thus this representation is what the algorithms will analyze; B algorithms merge all timelines together as described in [2]. The figures show the values of active metrics ($\neq 0$); if few are active, then they are shown in one timeline, otherwise, they are rendered individually to provide a better overview. For example, we can see that several metrics increase in Segment 12. We can also see an interesting result of our categorized coding, the write_bytes are bigger than 0 while write_calls are 0[5].

5 Evaluation

In the following, we assume the reference job (Job-M) is given, and we aim to identify similar jobs. For the reference job and each algorithm, we created CSV files with the computed similarity to all other jobs from our job pool (worth 203 days of production of Mistral). During this process, the runtime of the algorithm

[4] This can be support staff or a data center user that was executing the job.

[5] The reason is that a few write calls transfer many bytes; less than our 90%-quantile, therefore, write calls will be set to 0.

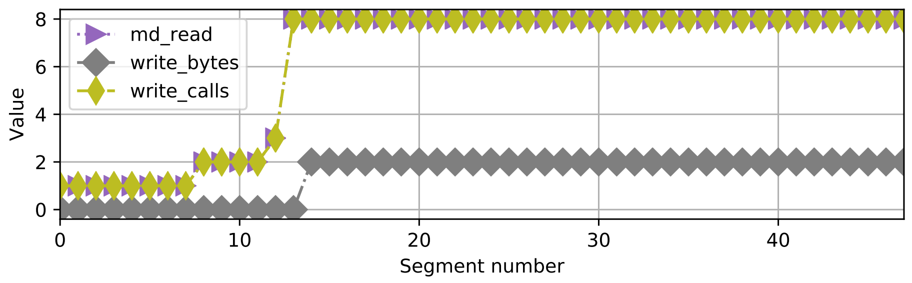


Fig. 1: Segmented timelines of Job-M (runtime=28,828 s, segments=48)

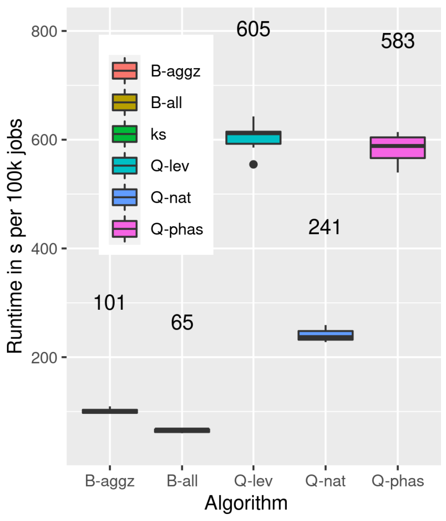
is recorded. Then we inspect the correlation between the similarity and number of found jobs. Finally, the quantitative behavior of the 100 most similar jobs is investigated.

5.1 Performance

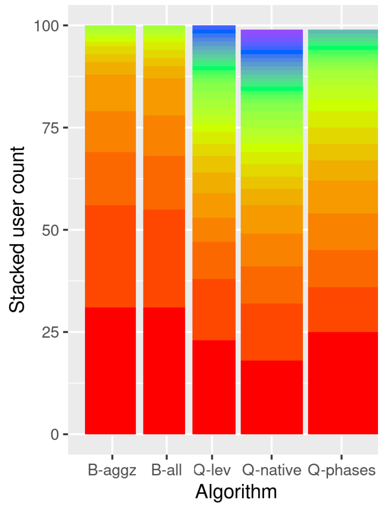
To measure the performance for computing the similarity to the reference job, the algorithms are executed 10 times on a compute node at DKRZ which is equipped with two Intel Xeon E5-2680v3 @2.50GHz and 64GB DDR4 RAM. A boxplot for the runtimes is shown in Figure 2a. The runtime is normalized for 100k jobs, i.e., for B-all it takes about 41 s to process 100k jobs out of the 500k total jobs that this algorithm will process. Generally, the B algorithms are the fastest, while the Q algorithms often take 4-5x as long. Q_phases and Levenshtein-based algorithms are significantly slower. Note that the current algorithms are sequential and executed on just one core. They could easily be parallelized, which would then allow an online analysis.

5.2 Quantitative Analysis

In the quantitative analysis, we explore the different algorithms how the similarity of our pool of jobs behaves to our reference job. The support team in a data center may have time to investigate the most similar jobs. Time for the analysis is typically bound, for instance, the team may analyze the 100 most similar jobs and rank them; we refer to them as the Top 100 jobs, and *Rank i* refers to the job that has the i-th highest similarity to the reference job – sometimes these values can be rather close together as we see in the histogram in Figure 3 for the actual number of jobs with a given similarity. As we focus on a feasible number of jobs, we crop it at 100 jobs (the total number of jobs is still given). It turns out that both B algorithms produce nearly identical histograms, and we omit one of them. In the figures, we can see again a different behavior of the algorithms depending on the reference job. We can see a cluster with jobs of higher similarity (for B-all and Q-native at a similarity of 75%). Generally, the growth in the relevant section is more steady. Practically, the support team would start



(a) Runtime of the algorithms to compute the similarity to our reference job



(b) User information for all 100 top-ranked jobs. Each color represents a specific user for the given data.

Fig. 2: Algorithm runtime and user distribution

with Rank 1 (most similar job, e.g., the reference job) and walk down until the jobs look different, or until a cluster of jobs with close similarity is analyzed.

Inclusivity and Specificity When analyzing the overall population of jobs executed on a system, we expect that some workloads are executed several times (with different inputs but with the same configuration) or are executed with slightly different configurations (e.g., node counts, timesteps). Thus, potentially our similarity analysis of the job population may just identify the re-execution of the same workload. Typically, the support staff would identify the re-execution of jobs by inspecting job names, which are user-defined generic strings.

To understand if the analysis is inclusive and identifies different applications, we use two approaches with our Top 100 jobs: We explore the distribution of users (and groups), runtime, and node count across jobs. The algorithms should include different users, node counts, and across runtime. To confirm the hypotheses presented, we analyzed the job metadata comparing job names which validate our quantitative results discussed in the following.

User distribution. To understand how the Top 100 are distributed across users, the data is grouped by user ID and counted. Figure 2b shows the stacked user information, where the lowest stack is the user with the most jobs and the topmost user in the stack has the smallest number of jobs. Jobs from 13 users are included; about 25% of jobs stem from the same user; Q-lev and Q-native include more users (29, 33, and 37, respectively) than the other three algorithms.

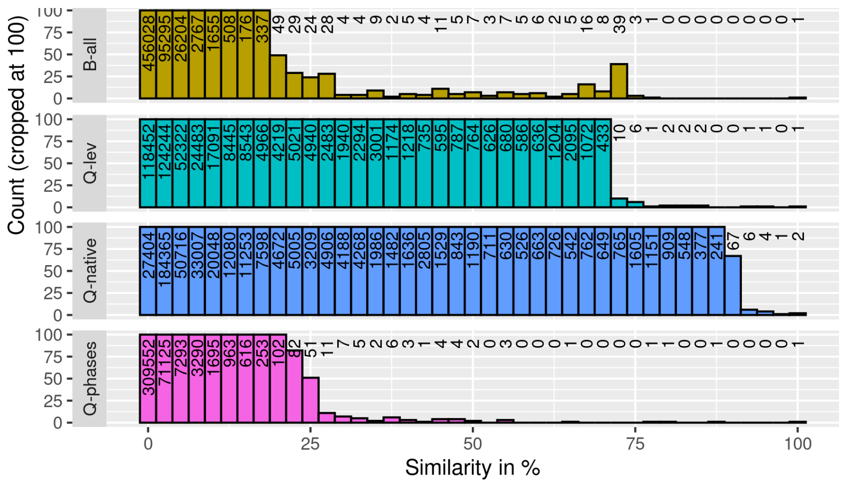


Fig. 3: Histogram for the number of jobs (bin width: 2.5%, numbers are the actual job counts). B-aggz is nearly identical to B-all and therefore omitted.

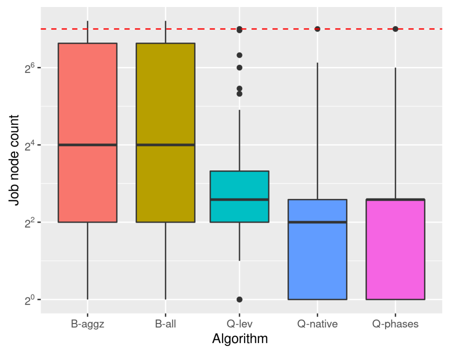
We didn't include the group analysis in the figure as user count and group ID are proportional, at most the number of users is 2x the number of groups. Thus, a user is likely from the same group and the number of groups is similar to the number of unique users.

Node distribution. Figure 4a shows a boxplot for the node counts in the Top 100 – the red line marks the reference job. All algorithms reduce over the node dimensions, therefore, we naturally expect a big inclusion across the node range as long as the average I/O behavior of the jobs is similar. We can observe that the range of nodes for similar jobs is between 1 and 128.

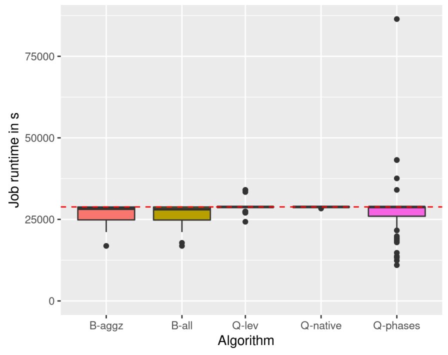
Runtime distribution. The job runtime of the Top 100 jobs is shown using boxplots in Figure 4b. While all algorithms can compute the similarity between jobs of different lengths, the B algorithms and Q-native penalize jobs of different lengths, preferring jobs of very similar lengths. Q-phases is able to identify much shorter or longer jobs.

6 Assessing Timelines for Similar Jobs

To verify the suitability of the similarity metrics, for each algorithm, we carefully investigated the timelines of each of the jobs in the Top 100. We subjectively found that the approach works very well and identifies suitable similar jobs. To demonstrate this, we include a selection of job timelines and selected interesting job profiles. Inspecting the Top 100 is highlighting the differences between the algorithms. All algorithms identify a diverse range of job names for this reference

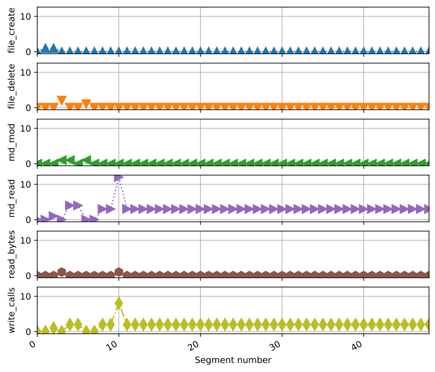


(a) Node counts ($job = 128 nodes$)

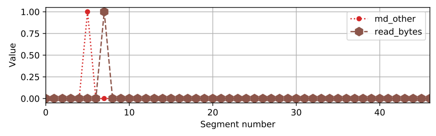


(b) Runtime ($job = 28,828s$)

Fig. 4: Distribution for all 100 top-ranked jobs



(a) Rank 15, SIM=73%

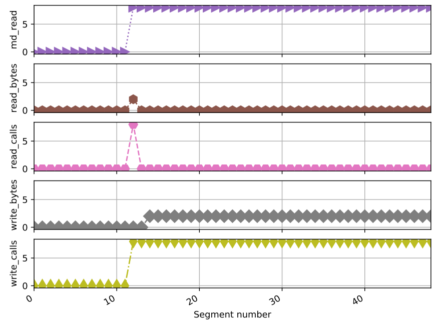


(b) Rank 2, SIM=78%

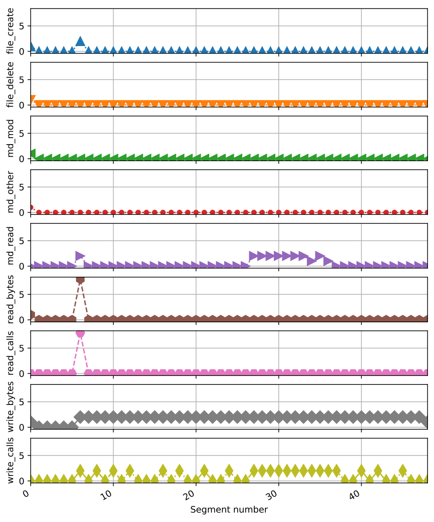
Fig. 5: Job-M with Bin-Aggzero, selection of similar jobs

job in the Top 100. The number of unique names is 19, 38, 49, and 51 for B-aggzero, Q-phases, Q-native, and Q-lev, respectively.

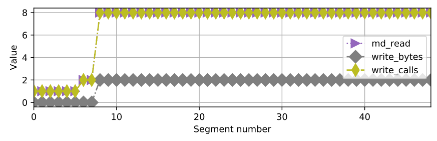
When inspecting their timelines, the jobs that are similar according to the B algorithms (see Figure 5) subjectively appear to us to be different. The reason lies in the definition of the B-* similarity, which aggregates all I/O statistics into one timeline. The other algorithms like Q-lev (Figure 6) and Q-native (Figure 7) seem to work as intended: While jobs exhibit short bursts of other active metrics even for low similarity, we can eyeball a relevant similarity particularly for Rank 2 and Rank 3 which have the high similarity of 90+%. For Rank 15 to Rank 100, with around 70% similarity, a partial match of the metrics is still given.



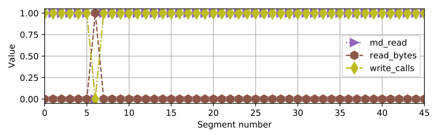
(a) Rank 3, SIM=94%



(b) Rank 15, SIM=74%



(c) Rank 2, SIM=95%



(d) Rank 100, SIM=70%

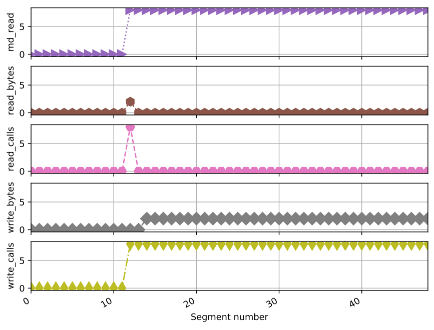
Fig. 6: Job-M with Q-lev, selection of similar jobs

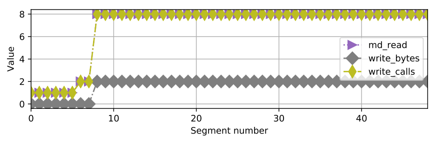
7 Conclusion

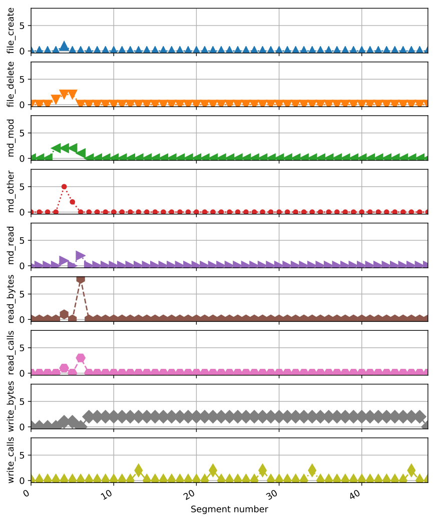
We introduced a methodology to identify similar jobs based on timelines of nine I/O statistics. The quantitative analysis shows that a diverse set of results can be found and that only a tiny subset of the 500k jobs is very similar to our reference job representing a typical HPC activity. The Q-lev and Q-native work best according to our subjective qualitative analysis. Related jobs stem from the same user/group and may have a related job name, but the approach was able to find other jobs as well. This was the first exploration of this methodology. In the future, we will expand the study by comparing more jobs in order to identify the suitability of the methodology.

References

[1] Jiwoo Bang et al. “HPC Workload Characterization Using Feature Selection and Clustering”. In: *Proceedings of the 3rd International Workshop on Systems and Network Telemetry and Analytics.* 2020, pp. 33–40.


(a) Rank 2, SIM=99%


(b) Rank 3, SIM=97%


(c) Rank 15, SIM=91%

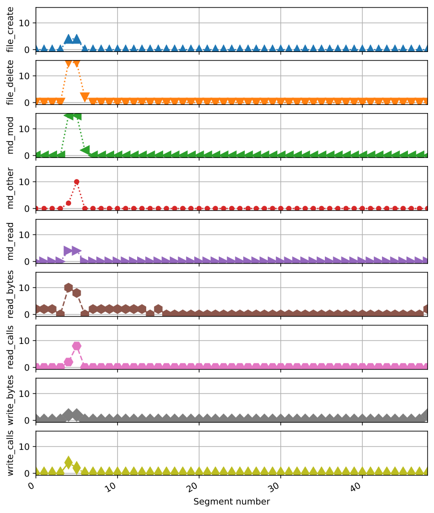

(d) Rank 100, SIM=88%

Fig. 7: Job-M with Q-native, selection of similar jobs

[2] Eugen Betke and Julian Kunkel. "Classifying Temporal Characteristics of Job I/O Using Machine Learning Techniques". In: *Journal of High Performance Computing*. Issue 1 (Jan. 2021). DOI: 10.5281/zenodo.4478960.

[3] Eugen Betke and Julian Kunkel. "The Importance of Temporal Behavior when Classifying Job IO Patterns Using Machine Learning Techniques". In: *High Performance Computing: ISC High Performance 2020 International Workshops, Revised Selected Papers*. Lecture Notes in Computer Science 12151. Springer, June 2020, pp. 191–205. ISBN: 978-3-030-59851-8.

[4] Nicolas Chan. "A Resource Utilization Analytics Platform Using Grafana and Telegraf for the Savio Supercluster". In: *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*. 2019, pp. 1–6.

[5] Orianna DeMasi, Taghrid Samak, and David H Bailey. "Identifying HPC codes via performance logs and machine learning". In: *Proceedings of the first workshop on Changing landscapes in HPC security*. 2013, pp. 23–30.

[6] Joseph Emeras et al. "Evalix: classification and prediction of job resource consumption on HPC platforms". In: *Job Scheduling Strategies for Parallel Processing*. Springer. 2015, pp. 102–122.

[7] Todd Evans et al. "Comprehensive resource use monitoring for HPC systems with TACC stats". In: *2014 First International Workshop on HPC User Support Tools*. IEEE. 2014, pp. 13–21.

[8] Mohamed S Halawa, Rebeca P Díaz Redondo, and Ana Fernández Vilas. "Unsupervised KPIs-Based Clustering of Jobs in HPC Data Centers". In: *Sensors* 20.15 (2020), p. 4111.

[9] Hassan Khotanlou and Amir Salarpour. "An Empirical Comparison of Distance Measures for Multivariate Time Series Clustering". In: *International Journal of Engineering* 31.2 (2018), pp. 250–262.

[10] Julian Kunkel et al. "Tools for Analyzing Parallel I/O". In: *High Performance Computing: ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected Papers*. Lecture Notes in Computer Science 11203. Springer, Jan. 2019, pp. 49–70. ISBN: 978-3-030-02465-9.

[11] Zhengchun Liu et al. "Characterization and identification of HPC applications at leadership computing facility". In: *Proceedings of the 34th ACM International Conference on Supercomputing*. 2020, pp. 1–12.

[12] Sandra Méndez et al. "A new approach for Analyzing I/O in parallel scientific applications". In: *Computer Science & Technology Series* (2012).

[13] Michael D Morse and Jignesh M Patel. "An efficient and accurate method for evaluating time series similarity". In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. 2007.

[14] Gonzalo Navarro. "A guided tour to approximate string matching". In: *ACM computing surveys (CSUR)* 33.1 (2001), pp. 31–88.

[15] Gonzalo P Rodrigo et al. "Towards understanding HPC users and systems: a NERSC case study". In: *Journal of Parallel and Distributed Computing* 111 (2018), pp. 206–221.

[16] Nikolay A Simakov et al. "A Workload Analysis of NSF's Innovative HPC Resources Using XDMoD". In: *arXiv preprint arXiv:1801.04306* (2018).

[17] Andrew Turner et al. "Analysis of parallel I/O use on the UK national supercomputing service, ARCHER using Cray's LASSi and EPCC SAFE". In: Oct. 2019.

[18] Matthias Weber et al. "Visual Comparison of Trace Files in Vampir". In: *Programming and Performance Visualization Tools*. Springer, 2017, pp. 105–121.

[19] Joseph P White et al. "Automatic Characterization of HPC Job Parallel Filesystem I/O Patterns". In: *Proceedings of the Practice and Experience on Advanced Research Computing*. 2018, pp. 1–8.