Information Fusion 93 (2023) 1–20

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full length article

A pipeline architecture for feature-based unsupervised clustering using multivariate time series from HPC jobs

Jonatan Enes [a,*], Roberto R. Expósito [a], José Fuentes [b], Javier López Cacheiro [b], Juan Touriño [a]

[a] Universidade da Coruña, CITIC, Computer Architecture Group, Campus de A Coruña, Spain
[b] Fundación Centro de Supercomputación de Galicia (CESGA), Santiago de Compostela, Spain

ARTICLE INFO

Dataset link: Datasets and source code for a pipeline architecture for feature-based unsupervised clustering using multivariate time series from HPC jobs (Original data)

Keywords:
Unsupervised clustering
Feature extraction
Multivariate time series
Anomaly detection
HPC jobs

ABSTRACT

Time series are key across industrial and research areas for their ability to model behaviour across time, making them ideal for a wide range of use cases such as event monitoring, trend prediction or anomaly detection. This is even more so due to the increasing monitoring capabilities in many areas, with the subsequent massive data generation. But it is also interesting to consider the potential of time series for Machine Learning processing, often fused with Big Data, to search for useful information and solve real-world problems. However, time series can be studied individually, representing a single entity or variable to be analysed, or in a grouped fashion, to study and represent a more complex entity or scenario. In this latter case we are dealing with multivariate time series, which usually imply different approaches when dealt with. In this paper, we present a pipeline architecture to process and cluster multiple groups of multivariate time series. To implement this, we apply a multi-process solution composed by a feature-based extraction stage, followed by a dimension reduction, and finally, several clustering algorithms. The pipeline is also highly configurable in terms of the stage techniques to be used, allowing to perform a search with several combinations for the most promising results. The pipeline has been experimentally applied to batches of HPC jobs from different users of a supercomputer, with the multivariate time series coming from the monitoring of several node resource metrics. The results show how it is possible to apply this multi-process information fusion to create different meaningful clusters from the batches, using only the time series, without any labelling information, thus being an unsupervised scenario. Optionally, the pipeline also supports an outlier detection stage to find and separate jobs that are radically different when compared to others on a dataset. These outliers can be removed for a better clustering, and later reviewed looking for anomalies, or if numerous, fed back to the pipeline to identify possible groupings. The results also include some outliers found in the experiments, as well as scenarios where they are clustered, or ignored and not removed at all. In addition, by leveraging Big Data technologies like Spark, the pipeline is proven to be scalable by working with up to hundreds of jobs and thousands of time series.

1. Introduction

The clustering and automatic processing of time series is a highly interesting topic if we take into consideration how time series are generated and used across a wide range of fields [1]. In addition, improvements in sensors, the rise of the Internet of Things, the emerging so-called Industry 4.0 and the advances in data storage and processing capabilities, have all opened the door to an increasing demand for time series related studies, and to new solutions that tackle challenging problems.

However, if we consider time series by their definition and try to apply most of the currently used Machine Learning (ML) techniques to extract valuable information from them, we face a challenge as time series are not single pieces of data, but rather a time-continuous and variable amount of usually numerical values. Even so, such challenge is rendered only more difficult if instead of individual, univariate time series, we try to apply ML to groups of related time series, that is, Multivariate Time Series (MTS), in particular when aiming for a clustering result. In this latter scenario, many well-described techniques in the literature [1] may fail to provide a solution as they are designed to work by analysing time series individually. On the other hand, characterizing a group of time series requires to somehow effectively summarize them or to select a subset of information that properly represents them as a group, an area of research in and of itself [2]. Furthermore, another issue arises for ML, and more specifically for clustering, when taking

* Corresponding author.
E-mail addresses: jonatan.enes@udc.es (J. Enes), rreye@udc.es (R.R. Expósito), jfuentes@cesga.es (J. Fuentes), jlopez@cesga.es (J.L. Cacheiro), juan@udc.es (J. Touriño).

https://doi.org/10.1016/j.inffus.2022.12.017
Received 22 January 2022; Received in revised form 14 December 2022; Accepted 16 December 2022
Available online 20 December 2022

into account the amount of data and the speed at which it is generated. This makes the use of supervised learning and data labelling quite tricky, thus promoting the use of unsupervised approaches instead.

In this paper, we present a new solution in the form of a pipeline that deals with this complex scenario, and looks for a flexible, yet simple approach of clustering MTS using both ML techniques and Big Data technologies in a scalable and seamlessly fused manner. The novelty of our approach is thus centred on two fronts:

- The technical one, with a focus on the design and implementation of a scalable and efficient pipeline, used for unsupervised ML clustering, that leverages Big Data technologies.
- The ML one, with a focus on the pipeline design and strategies for a flexible and adaptive unsupervised clustering of MTS using a feature-based approach. In addition, the pipeline can be used to detect outliers, which could potentially be anomalies, and to apply clustering to them if deemed interesting.

More specifically to our scenario, the groups of time series, or MTS, correspond to High Performance Computing (HPC) jobs executed on a real supercomputer, the Finis Terrae II hosted at the Galicia Supercomputing Centre (CESGA) [3]. These time series are extracted from measurements of different resources obtained from monitoring the infrastructure. Each job spans several reserved and dedicated computing nodes, which are in turn monitored for different resources such as CPU and memory. The fact that numerous jobs are continuously executed on this HPC environment explains why data cannot be properly tagged, or at least not in a useful way, thus being a potential candidate for unsupervised clustering.

Multiple experiments presenting different scenarios are proposed, using as input datasets that contain jobs extracted from this supercomputer: (1) jobs executed by a single user; (2) jobs from the pairwise combination of previous single users; and (3) jobs from multiple users selected using a time window. The results for all these experiments are analysed, in some cases also including the outcome of applying outlier detection and clustering. In addition, and considering the unsupervised nature of our scenario, we extensively back the experimental results with several visualization techniques which prove useful in guiding their human interpretation, specially when the separation of the clusters requires many dimensions. The use cases exposed by the experiments can be useful to system administrators to identify groups of jobs with similar resource usages and extract job patterns. This can help guiding decision-making processes that involve the HPC infrastructure, like defining separate policies for different job types. Furthermore, the outlier detection can be used by both system administrators and HPC users to find potential anomalies (e.g., a job with a different resource pattern), which once isolated can be the subject of study. In addition, these anomalies, if numerous, can be clustered to measure the similarity between them, or even to find groups of closely related anomalies (e.g., a misconfigured job that keeps failing, or a computing node with failing hardware).

The rest of the paper is structured as follows. Section 2 briefly comments on the current state of the art about time series classification for the supervised use cases, and clustering for the unsupervised scenarios. Current research that focuses on anomaly detection is also discussed. Section 3 focuses on the design of the proposed pipeline architecture. It first introduces some key terminology, continues by describing the main objectives, technical details and experimental configuration of the pipeline stages, and finishes with a discussion about scalability and the parallelization approach. Section 4 presents several experiments backed both by using evaluation information, as well as visualization of the actual resource plots and the clustering. In addition, an analysis of the computational complexity is described using time measurements of the experiments. Finally, Section 5 presents the main conclusions of the paper.

2. Related work

Considering that time series analysis is an interesting topic, there are current approaches that can be used to extract useful information from them via grouping techniques, either using supervised or unsupervised learning, and also for the outlier and anomaly detection use cases. Next, we tackle both aspects separately in Sections 2.1 and 2.2, respectively.

2.1. Time series classification and clustering

One of the most basic ways of extracting knowledge from time series is by merely grouping them, a task that can be performed using either classification in a supervised ML approach, or with clustering and an unsupervised approach. On the one hand, classification techniques use previously available and external information of the time series to be analysed in order to better characterize the existing groups or even create them. This in turn allows to study unclassified time series or data and match them to an existing class, and if a group or class is missing, create a new class. On the other hand, if there is no previous information, we have to resort to applying clustering, where groups of time series are identified only by their intrinsic similarity to each other. This unsupervised ML approach can also be seen as semi-supervised, if at any point some kind of metadata or human intervention is employed for evaluation purposes.

Regarding supervised learning, there are multiple recent works that expose different approaches to create the class groups using their time series as input examples. Even though this work does not focus on any supervised technique, they are still interesting to review as the tools and technologies used may be the same, and they offer a basis for comparison purposes. For example, Convolutional Neural Networks (CNNs) are used in [4] to choose characteristics to summarize and then classify time series. Such CNNs have also been employed in [5] for the classification of MTS heavily focused on an industry use case, and in [6] in conjunction with Dynamic Time Warping (DTW). DTW is a well-known and widely studied approach that uses point-to-point similarity measurements to calculate distances between series. For instance, authors in [7] focus on revisiting this technique and improving on it using ensembles. In the case of MTS, [8] explores how DTW can be applied in such scenarios. Other works use the shapelet-based or subsequence-based approach, which only takes into account relatively small portions of the time series, usually the ones that carry the most information or allow to better identify them. Shapelets have also been widely studied as they have proven to be reliable to characterize time series, albeit this technique is usually restricted to short or more or less predictable time series. Besides shapelets, the approach in COTE [9] combines several additional techniques in order to train different models and create an ensemble of predictors. The techniques used in COTE rely on four approaches, the time and frequency domains and the change and shapelet analysis. From this work it is interesting to note the idea of using an ensemble of predictors to build a more robust and flexible approach, relying on several models rather than on a single one. In [10], the authors take a step further from COTE and improve on the efficiency of the shapelet-based technique with the aim of reducing the computational complexity of the base algorithm. It is worth mentioning that this algorithm is implemented with Big Data technologies (i.e., Apache Spark [11]) to improve its performance. This in turn proves how these technologies can offer a reliable basis to build scalable solutions. When it comes to MTS and shapelets, some works like [12] have succeeded in extending the shapelet-based analysis approach in such scenarios. Another approach, probably popular when dealing with high amounts of data to be processed, is to summarize the time series in a limited set of measurements or characteristics that retain as much information as possible while also dramatically reducing dimensionality. For this type of feature-based clustering, several works propose different techniques and sets of characteristics to better cluster and summarize time series. Works like [13] use this approach for univariate time series, while

others like [14] extend this technique for multivariate time series. A last approach close to the feature-based ones is proposed in [15], which uses an elaborate fingerprinting method that is able to create a good hierarchical classification of time series even when they present noise.

All these previous works prove that when previous information is available, the use of classification is highly recommended, as the guidance of the existing classes not only potentially improves the results but may also grant a degree of validation of such results. However, it is worth commenting on some of the limitations they face. Some approaches like DTW and the shapelet-based one are sensitive to the length of the time series. So, applying them on scenarios where time series can potentially have radically different duration may be difficult. In the case of DTW, it usually requires knowing beforehand the shapes we are looking for. Other solutions can have difficulties when scaling to encompass large amounts of time series, or MTS with a non-defined amount of such time series.

However, an even major limitation is present on those scenarios where no external information, metadata or previously identified classes are available. This lack of knowledge may arise for many reasons such as the unfeasibility of tagging or documenting a large amount of data, which may even be generated continuously and automatically. In these scenarios, there are still two options available. On the one hand, it is possible to create or infer labels from the raw data in order to turn the scenario into a supervised one, as proven by works like [16]. And on the other hand, it may still be possible to extract information to some degree without depending on any external knowledge of any kind or using any sort of label information, thus ultimately falling back to unsupervised time series clustering.

When it comes to unsupervised time series clustering, there is less research available for new techniques, considering the limited amount of procedures that can be applied, specially for MTS and in the absence of labelled data or assigned classes, as described in [17]. Considering the approaches presented by these authors, some works have been able to perform clustering using only raw time series based on distance calculation like DTW [18], distance-based fuzzy clustering [19], shapelets [20], or even using visual recognition applied to time series [21]. We will however leave aside both raw data-based and model-based approaches to focus on feature-based ones. The reason behind is to increase flexibility regarding the data to be used, and to improve the chances of successful clustering across a wide range of scenarios, both with long and short time series, and with datasets containing jobs with unequal number of resource time series or even unequal in terms of length. In addition, we aim at avoiding having to look for any specific shapelet or subsequence inside the time series, which would be incompatible with a scenario of long and numerous time series, as well as limited human supervision and intervention available.

Regarding the feature-based approaches, there is a lot of research that seeks to come up with good feature-based representations of time series, some of which are applied to clustering. Such works usually aim to improve the classical Symbolic Aggregate approXimation (SAX), either through statistical features [22] or trend analysis [23], among others [24]. Unfortunately, the current state of the art regarding feature-based or symbolic-based clustering mostly leaves aside MTS when features have to be extracted. Some works that take into account MTS for unsupervised clustering, as well as a similar methodology as the one we used, include [25,26]. On the one hand, the first work looks for the best combination of feature extraction from highly multidimensional sensor data and subsequent clustering, in a scenario involving test and real driving situations, and where the generated clusters must always contain samples of both situations. This latter requirement is ultimately the condition that the clustering results must meet. On the other hand, unsupervised clustering of HPC jobs was carried out in the second work with the objective of finding both the best resource metrics to separate the jobs, as well as the best method and its parameters. The results from these two previous works prove that the combination of feature extraction with a dimension reduction and then clustering, using both hierarchical and centroid-based algorithms, should provide good results. These works also highlight that the major issue is the actual finding of the best predictor, and specially choosing the parameterization for the dimension reduction and clustering models. In addition, the methods and configuration used may depend on the input data. Nonetheless, our approach is closely related with [26], although we provide a more flexible and scalable design and implementation. Moreover, we extensively use visualization plots of the clustering results, which greatly help to understand and further back the predictions, as well as being significantly more human-friendly. And on this last point it is worth mentioning [27], which applies a semi-supervised feature-based clustering process intended to be flexible and scalable, but also offering the user the chance to view and alter the progress of the stages as needed.

In this paper, we present a pipeline that extracts information from HPC jobs by creating groups. The pipeline computes several functions to summarize an MTS into a set of features, which are then used to optionally extract outliers, perform several clustering predictions, and finally evaluate and rank such predictions according to their expected accuracy. Furthermore, this pipeline is applied to untagged time series with little to no metadata or class information from them, so we rely solely on unsupervised learning for the clustering.

Finally, it is also worth noting some other works that have focused on exploring the best ways of characterizing time series through libraries and functions. Among other models for selecting features, we can emphasize [28] like one of the most extensive options, obtaining more than 7000 parameters by each series. On the other hand, packages like Tsfresh [29] and catch22 [30] offer a smaller set but include the essential features for most problems.

2.2. Outlier and anomaly detection

As previously mentioned, the analysis and classification of time series data is of great interest because of their ubiquity, flexibility and potential application in all kind of areas that generate time series, including industry, HPC, health or biological research. However, many of such scenarios produce ‘raw’ time series, that is, they may produce data in a fashion that makes it difficult to be labelled, thus having to resort to unsupervised time series clustering. Such clustering may prove interesting not only to look for specific classes or groups which can be later analysed, but also to look for specific patterns correlated to interesting or sought-after information. There are several examples in the literature such as detecting inefficient applications on a supercomputer [31], genetic similarity [28], or anomalies on sensor measurements [32], among others. These use cases can be considered an example of looking for the abnormal and outlier data among a deluge of ordinary time series, or what is the same in some scenarios, detecting anomalies.

In the context of outlier and anomaly detection, time series data can also be analysed to detect out-of-the-common behaviour. For example, the authors in [33] propose finding change points or discontinuities to identify anomalies. In [18], DTW is used to identify anomalous time series data. Other works such as [34] use autoencoders in order to establish a ‘normal’ status of an HPC system, and then compare anomalous data to such normality, or even graph-based approaches after a proper transformation of time series to graphs and a feature extraction of such graphs [35]. Another option proven in several works involves clustering, which can be used to identify tiny, or even single-element clusters, as clusters that contain anomalous data. In some cases, these data can be identified as elements hard to assign to any cluster, which is natural to some density-based clustering algorithms such as DBSCAN [36]. It is also possible to use the Long Short-Term Memory (LSTM) [31] technique for feature extraction and classification of anomalous jobs on a supercomputer, even being able to do online classification. Outside of deep learning, we can highlight the

work described in [37], where Gaussian mixed models are used to classify computing jobs and detect anomalies using up to three resource metrics.

In our scenario, however, we rely on an outlier-based approach for job anomaly detection, where the outliers are extracted from the combined results of two outlier detection methods performed at an early stage, before any actual clustering is carried out. These outliers are chosen because they are too far apart from their closest neighbours according to the distances computed from the features extracted. If not extracted, these outliers may deteriorate the results as they may force the following clustering stages to place them on small clusters of their own, sometimes even having to create isolated single-job clusters. Once extracted, and specially if the number of outliers is high and we suspect that there could be groups of jobs with a similar anomaly pattern, it is also possible to refeed the pipeline with a dataset containing only these outliers and starting anew from the beginning. Overall, this outlier-based detection using several detectors and combined knowledge has already been successfully explored as part of the ensemble learning [38], although our approach is less sophisticated and more conservative, mainly aiming at detecting potential outliers and separating them from the main data.

3. Pipeline architecture for time series clustering

To achieve the goals previously laid out in Section 1, a multi-process information fusion pipeline has been designed and implemented from scratch. This pipeline has been divided into different stages, each using specific technologies and with its own objectives for information processing. Overall, the pipeline starts the processing with a basic input file containing job IDs, proceeds through the different stages and finally produces a result that consists of several clustering predictors, which are also in turn ranked using evaluation metrics. This result can be visualized through several tools and techniques. However, the technologies used to implement and efficiently execute these stages come from different areas, mainly from Big Data and ML, and have been fused to work together. First, Section 3.1 introduces some relevant key terms used throughout the paper, in order to next explain in detail each pipeline stage in Section 3.2. Finally, Section 3.3 provides an analysis of the pipeline's scalability and computational complexity, as well as some details regarding the parallelization techniques used.

3.1. Previous concepts

The pipeline that lies at the core of our proposal has been specifically designed to follow the general recommendation for ML pipelines, that is, to focus on transformers. According to the widely used Scikit-Learn library [39], the definition of transformers in this context would be of stateless functions or procedures that implement a basic 'transform' operation, which mainly consists of taking a set of data as input and returning an updated set, either modifying the existing one or adding new information. Such design pattern, which is described in [40] and extensively used in research works based on pipeline architectures [41,42], allows the stages of the pipeline to be flexible and reusable, as the transformers that implement them can be chosen on demand and chained in a specific order as needed, while remaining decoupled from the data being processed. This decoupling is of special interest when considering that the pipeline is going to process unsupervised data, that is, when the properties of the data may be unknown and some trial-and-error experimentation might even be expected. Moreover, this design enhances parallelism, as transformers can be applied in parallel as long as their inputs and outputs are not dependent on each other.

In our pipeline, specific transformers have been created according to the objectives of each stage, as they may have slight differences (e.g., feature extraction transformer, clustering transformer). Nevertheless, they expose the same 'transform' operation and common basic input/output parameters in order to be executed in an abstracted way. Python has been used to implement the transformers and any auxiliary code, relying on Pandas DataFrames [43] as the basic table data structure. Each transformer applies an operation to a subset of a DataFrame and changes it accordingly, usually by adding new columns. In addition, such DataFrames can be persisted as files if needed using the HDF5 file format and can be seamlessly serialized and deserialized.

When it comes to the data that is processed, time series come from readily available and measurable resource metrics such as User CPU and Cached memory. We will refer to these metrics in a grouped way (e.g., CPU, memory) throughout the paper according to the resource they are based on, as the metrics within a group are used either all of them or none of them. However, considering that raw time series cannot be directly or easily used for clustering, a processing stage is required to extract features in the form of discrete values, which on the other hand are readily usable.

Finally, it is important to define what a 'job' is, because this work has been centred around job clustering on an HPC system. In our scenario, a job represents a user's workload executed in a specific time window and using a set of dedicated computing nodes. Considering this multiple node execution, each job will have several time series, even for the same resource metric, as each node is individually monitored. On a last note regarding jobs, although it is usually possible to extract metadata from the databases provided by the queue manager/job scheduler of the HPC system, no additional information has been used aside from the user identifiers, which are only required to aid in assessing the results of experiments where multiple users are present.

3.2. Auxiliary and pipeline stages

Overall, a total of eight stages have been designed and implemented, not only from the point of view of clustering with their specific purposes, but also from the point of view of their execution considering that they have different computational requirements, as further explained in Section 3.3. As can be seen in Fig. 1, the stages can be classified into those that are auxiliary and those that actually create the pipeline. There are also some other processes involved, like those part of the infrastructure operation, such as the *Job Scheduling* and the *Resource Monitoring*, which will only be briefly mentioned when relevant to this work, considering that they are mostly external; or the *Visualization* process, which will be extensively used for the experimental results discussion. Regarding the three auxiliary stages (*Job Retrieval*, *Data Collection* and *Preprocessing*), their task is to prepare the data to be processed on the actual pipeline. First, the *Job Retrieval* stage produces a list of job IDs. Afterwards, the *Data Collection* stage retrieves the basic required metadata (e.g., node list). And finally, the *Preprocessing* stage retrieves the resource time series and combines them with the basic job metadata, creating a file with all the required information that will be later used in the pipeline stages. These auxiliary stages do not follow a transformation-based design and because of this they are not strictly a part of the ML pipeline, although they are still required. Regarding the five pipeline stages (*Feature Extraction*, *Outlier Detection*, *Dimension Reduction*, *Clustering* and *Evaluation*), they are executed in that order to produce the final output, although the *Outlier Detection* is optional. It is also interesting to note that they can be executed either in a continuous fashion, or stage by stage using intermediary files. Furthermore, some of these stages (i.e., the most demanding ones) can be parallelized to improve performance, considering that they perform the bulk of the processing, as detailed in Section 3.3.

Next, each stage is thoroughly described detailing its objective, its implementation when necessary, the configuration parameters available for that stage and their experimentally used values. To aid in this purpose and given the extensive number of parameters, Table 1 summarizes such configuration.

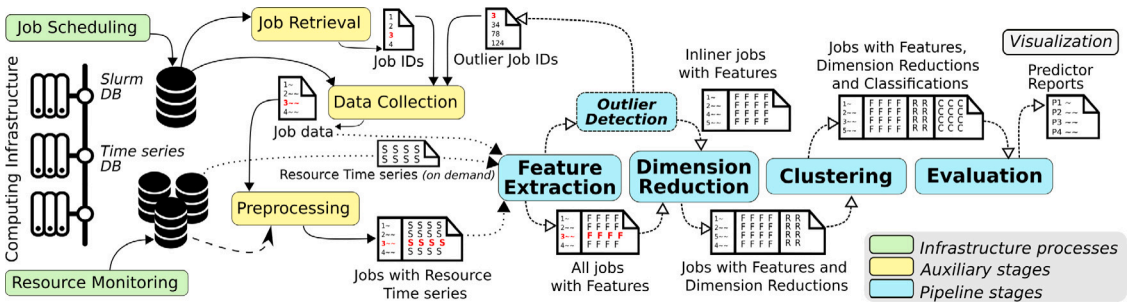


Fig. 1. High-level overview of the clustering pipeline architecture.

Table 1
Configuration of all the stage parameters with the experimental values used.

Stage	Configuration parameters	Experimentally used value
Job Retrieval	• time window • minimum number of job nodes • minimum job runtime	• 2 UNIX timestamps • 3 nodes • 90 min
Preprocessing	• time series downsampling • resource metrics	• 3 min • CPU and memory
Feature Extraction	• extraction functions • aggregation functions	• Several statistics and Tsfresh functions (Table 2) • max, q75, mean, q25, min, std, skew
Outlier Detection	• # neighbours • % of features • % of samples	• 5% of dataset job number • 50% of features • 50% of samples
Dimension Reduction	• dimension reduction algorithms	• Kernel PCA, Spark PCA
Clustering	• clustering algorithms	• KMeans, AgglomerativeClustering
Evaluation	• evaluation functions	• Silhouette, Davies–Bouldin, Calinski–Harabasz, number of clusters, cluster histogram kurtosis and skewness

3.2.1. Job retrieval

The first task is to select which jobs will be the subject of processing, and create a job ID list. This task, carried out by the *Job Retrieval* stage, consists of a database query in search of those jobs that match the user's criteria, such as a minimum job runtime, or even specific user IDs. These metadata, which are continuously created and updated by the *Job Scheduling* process, are stored in a relational MySQL database, being commonly used by the Slurm scheduler [44] for job management tasks or subsequent analysis. In our scenario, we use two timestamps to specify a *time window*, a *minimum number of job nodes* and a *minimum job runtime* as parameters to extract datasets from the database (see Table 1). Ignoring the timestamp values, which by themselves are not important, we set a minimum of 3 nodes and 90 minutes of job runtime to be eligible. With these values we aim at discarding jobs that are either too simple or too short for a supercomputer. The inclusion of these jobs in the pipeline would not be beneficial, as with a very low amount of time series points some feature extraction functions will only produce unusable values (e.g., Infinite), or if the number of nodes is too low there is a high probability of having radically different resource patterns due to the frequent use of a manager/worker job configuration in HPC workloads.

3.2.2. Data collection

After the jobs are retrieved and listed, it is necessary to collect basic information about them, a task carried out by the *Data Collection* stage. The details of each job (e.g., node list, job start and finish timestamps) are extracted from the same metadata database. Using such timestamps it will be later possible to query for the resource time series of the job, and more specifically, of each used node from the node list. Considering that this is a simple stage that cannot be configured, it has been omitted in Table 1.

3.2.3. Preprocessing

If the pipeline is used to perform batch processing, that is, with previously selected and listed jobs such as in our scenario, it may be interesting to run an initial preprocessing stage that creates a file where both the job metadata and its resource time series are combined. Even though optional, this stage is useful when different pipeline configurations are intended to be used, as in such scenario the time series retrieval, which may be costly in terms of time, is only performed once and not for every pipeline execution. If this stage is skipped, or if there is a need to retrieve time series as they are generated (i.e., while the jobs are being executed), they can also be retrieved on demand in the *Feature Extraction* stage.

The resource time series are continuously recorded by various monitoring agents deployed on the computing nodes, including telegram and collectd [45], as part of the *Resource Monitoring* infrastructure process, and ultimately persisted on a time series database deployed using OpenTSDB [46] and HBase [47]. This database solution provides an environment where data can be stored without scalability issues, in order to be later queried by either this stage or the *Feature Extraction* one.

When it comes to the monitoring and its configuration, it is performed using a certain sampling frequency, an important parameter that sets the degree of granularity of the time series. In our case, the monitoring agents have been configured by the system administrator to take samples every minute (i.e., the time series can be as fine-grained as one minute). Even when considering that this sampling frequency may be enough for many use cases, it is worth mentioning that time series can still be easily downsampled using the mean value, something that is performed on the fly at query time. This downsampling value can be set according to the more or less expected job runtimes present in the dataset, although the results should not vary greatly provided that the values used are sensible. While high downsampling values will inherently cause a loss of information, low values may cause longer

Table 2
Extraction functions and their configuration parameters.

Library	Function	Parameters
Statistics	max q75 mean q25 min std skew	None

Library	Function	Parameters	
Tsfresh	count above mean count below mean first location of maximum first location of minimum abs energy absolute sum of changes kurtosis	None	
	agg autocorrelation	• aggregate function	• max lag
	augmented dickey fuller	• attribute	• autolag
	c3	• lag	
	cid ce	• normalize	
	ratio beyond r sigma	• r sigma	
	fft coefficient	• num coefficients	
	agg linear trend	• chunk length	• aggregation function
	ar coefficient	• coefficients	• delay samples

processing times for the *Feature Extraction* stage. In our scenario, a value of 3 min was used for the *time series downsampling* configuration parameter (see Table 1).

Regarding the resource metrics themselves, there are many different ones collected by the agents from the hardware infrastructure that can be effectively used for the clustering: from metrics specific to the computing nodes such as CPU, memory or disk, to other metrics involving hardware monitoring such as temperature or fan speed. Nevertheless, we will focus on two crucial resources, CPU and memory, selected using the *resource metrics* parameter as shown in Table 1. These metrics are well suited to characterize a wide range of applications and patterns.

3.2.4. Feature extraction

Considering that in our approach we intend to perform clustering of jobs based on a set of features extracted from their time series, this stage is at the core of the pipeline by transforming the time series (i.e., a series of values), which may not be suitable for the distance calculation between different jobs as needed by the clustering algorithms, into discrete features (i.e., single values). This conversion allows using the data in a more straightforward manner. This is referred to as feature extraction, as time series are processed to 'extract' numerical values that best represent them. To achieve this, mathematical functions are used that ideally return different values for time series with varying trends, and similar values for those that expose a similar behaviour. This will allow to tell apart different time series, and to cluster those that are similar or even identical.

There are diverse *extraction functions* of different types that can be used to 'summarize' time series. Most of them are provided by libraries such as Tsfresh or catch22, although we also opted to implement some basic summary statistic functions (see top of Table 2). Overall, these functions can be as simple as reporting the maximum and minimum values of a time series, functions of medium complexity such as counting the number of points above or below the mean, or of high complexity such as obtaining Fourier coefficients or entropy values. In addition, some functions can be parameterized with different values, usually the more complex ones like those provided by the Tsfresh library. In order to adhere to our design of a pipeline based on transformers as previously explained in Section 3.1, every parameterized function counts as one transformer (i.e., the same function with different parameters produces several transformers). All the functions used in this work are listed in Table 2, as named in the Tsfresh library [48], together with their configuration parameters (one parameter per bullet).

However, because jobs are composed of several nodes (i.e., several time series), even if features are extracted from the time series, the result is that for each metric and extraction function there are multiple values, one per computing node. This would not be a problem if the number of values (i.e., the number of nodes) was the same for all the jobs. As this fact cannot be ensured (the number of nodes is chosen by the user at job submission), we are forced to further reduce such variable number of feature values down to a single one. This additional step is implemented by the *aggregation functions*, which take several values and return just one by performing an aggregation operation, also commonly known as a reduction operation. The aggregators used in our scenario are seven, two that perform a selection, both the maximum and minimum values, and five that perform a computation: the 25 and 75 quartiles, mean, standard deviation and skewness values. In addition, each aggregator is applied to each existing feature by using a transformer, in a similar way as with the extraction functions. This reduction step is critical to later apply clustering algorithms, as it allows working with structured data.

Overall, the combination of extraction and aggregation functions implement the *Feature Extraction* stage by taking a group of time series for a resource metric, extracting the features and then reducing the node feature values down to a few ones using aggregation operations. This procedure is depicted in Fig. 2. Taking Job 2 as an example, which was executed with 4 nodes, there is a total of 8 time series, 4 for every metric of the 2 available ($M1$ and $M2$). Next, we can we apply 2 feature extraction functions ($Fe1$ and $Fe2$) and the result will be 16 values, one for each combination of time series and feature extraction function. As such number still depends on the number of nodes, we then apply 2 aggregation functions ($A1$ and $A2$) so that the previous results are reduced to only 8, one for each combination of metric (2), feature extraction function (2) and aggregation function (2). Therefore, the dependency on the number of nodes has been removed in this last step. Basically, this two-step procedure structures the data and removes such dependency, as Jobs 1 and 3 also have the same number of results at the end of the stage, even though they started with 3 and 5 time series for every metric, respectively.

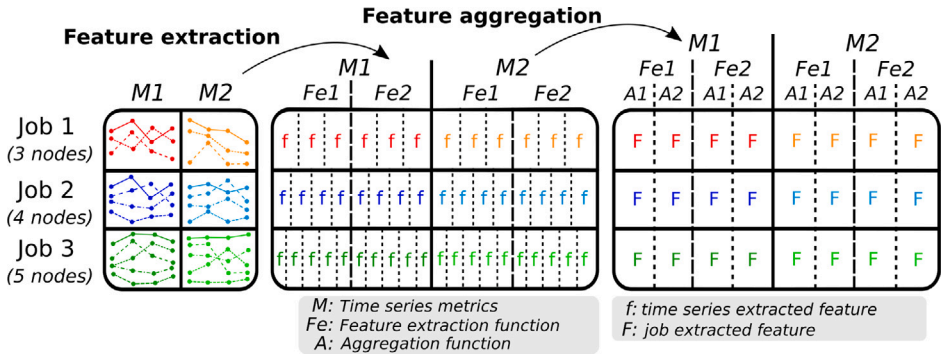


Fig. 2. Example of the feature extraction stage for 2 metrics, 2 extraction functions and 2 aggregators.

However, the extraction functions might produce invalid values (e.g., Not Available, Not a Number, Infinite) depending on the data, specially for some parameters or for time series with 'undesired' characteristics (e.g., too stable, short). This forces to perform a final 'invalid value' removal step, as these values cannot be carried through the pipeline. In our scenario, any extracted feature that has any invalid value for any job is removed. Although it may cause a loss of potentially useful data in most of the jobs, even when just a few are affected, it prevents any failure down the pipeline in the end. It can be argued that such invalid-value generating functions or parameters may be removed, but we think that it is better to keep them in the *Feature Extraction* stage for flexibility reasons. It is not entirely possible to know beforehand which functions or parameters will produce the most useful or invalid values, and it would also require human intervention to decide which of those functions and parameters should be kept and which removed.

Finally, in order to carry out this stage, the transformers are executed using Spark as the complexity of this stage is very high, as will be explained later in Section 3.3. With Spark it is possible to partition the data and parallelize the execution of the stage.

3.2.5. Outlier detection

With a set of features extracted, it is possible to optionally carry out an outlier detection stage that identifies those jobs that are radically different when compared to the dataset as a whole. Although not strictly necessary, this stage may be interesting for two reasons. On the one hand, these outlier jobs are removed from the larger dataset (inliner jobs) to help the next stages in achieving a better clustering. This is considering that a few outlier jobs may significantly shift the clustering and thus increase the difficulty of choosing a good prediction on an unsupervised scenario, where we do not know if a job or group of jobs can be safely ignored to perform a better classification which avoids overfitting. On the other hand, we can still extract valuable information from these outliers, both as potential isolated anomalies, and as the source of a dataset of outlier jobs that can also be the subject of clustering in the search of groups of related patterns.

In our scenario we implement this stage using two different outlier detection methods, LocalOutlierFactor and IsolationForest, which are available in the ScikitLearn library. These methods are combined in order to have some balance across different scenarios, considering that one method might detect an abnormally high number of outliers in some situations while the other only isolates a few, or vice versa. Ultimately, we opted for a conservative approach and only those outliers detected by both methods are chosen and removed from the dataset. When it comes to their configuration, LocalOutlierFactor is configured with the number of closest neighbours whose distance is used to compute the outlier score for each sample, and IsolationForest is configured with the number of features to create the forest trees and the number of samples to create the partitions. The experimental values used are 5% the number of jobs to determine the number of neighbours (e.g., 7 for a dataset of 130 jobs), and 50% of both the features and jobs for the forest trees and samples (see Table 1). Finally, these outliers can also be ordered according to a final score computed by adding up the normalized scores from the two predictors, and again normalizing such result.

3.2.6. Dimension reduction

Once the features have been extracted and properly structured, either with or without outliers, the end result can be of several hundreds of columns, mainly due to the combinatorial explosion of extraction and aggregation functions. So, it is essential to reduce the dimensions of this feature matrix (i.e., the 'search space') before the clustering takes place. This is backed by the high probability of having a large number of these columns with a very low variability in terms of values, that is, of being unusable to differentiate a group of jobs from another one. Unfortunately, this depends on the dataset and thus we cannot know beforehand which will be the most useful or useless features, that is, which columns will present the most variability. Moreover, the remaining features can have highly variable value ranges, considering that the input resource metrics can include very low values in the range of hundreds (e.g., User CPU) or extremely high ones in the range of thousands (e.g., Cached memory).

In order to address all these issues, it is useful to apply first a feature scaling and a dimension reduction afterwards. On the one hand, this stage uses a scaler that removes the mean and scales to unit variance for each feature column. On the other hand, it uses several *dimension reduction algorithms* that look for the features that hold the most amount of variability (i.e., the set of columns that can be used to better separate the jobs). With the chosen features, a reduced set of coordinates is created that may hold +90% of the variation of the original search space.

For the implementation of this stage, our pipeline uses the following methods provided by the ScikitLearn library: StandardScaler for the scaling and Principal Component Analysis (Kernel PCA) for the dimension reduction algorithms, as well as a Spark-based Python implementation of a linear PCA (Spark PCA) [49]. As with the feature extraction functions, each algorithm can be parameterized and integrated in our pipeline as a transformer. The main parameter is the number of components to use for Kernel PCA, a value we safely set to 100, or the minimum variance to look for with Spark PCA, which in our case is 90%. In addition, it is possible to apply Kernel PCA with a linear reduction, or a second or third degree polynomial reduction.

The result of this stage is a projection of the previous dimensions into fewer new ones, so throughout the paper such result may also be referred to as 'projection', and the resulting dimensions as 'components'. Note that besides creating these projections, it is also possible to know in this stage the variance captured by each component, and its mathematical composition, by using the linear PCA reductors specifically. Two pieces of useful information can be extracted from these data: (1) the number of components needed to capture a percentage of variance, which may hint at the degree of variability of the dataset;

and (2) which resource metrics were more important in determining the projection, derived from the mathematical composition of the features and considering that these were parameterized using metrics. Although this information may not always fully apply (e.g., if a polynomial reduction is used), it may aid the human in case the results are inspected, particularly in picking the resource plots to view. Such information will be later used for some of the experiments presented in Section 4.

3.2.7. Clustering

After the features have been extracted and a small set of dimensions, or projection, has been created from them, it is now possible to apply several clustering models. Although many algorithms can be used for the clustering of the jobs, we opted for distance-based ones, whether they are centroid-based (e.g., KMeans) or hierarchical (e.g., Agglomerative). This decision comes naturally considering that the components used are purely numerical.

For these *clustering algorithms*, we rely on KMeans and AgglomerativeClustering as provided by ScikitLearn. Both can be parameterized with the number of clusters to use (typically, the 'k' parameter), which in our case varies from 2 to 30 clusters in steps of 1. This upper value has been chosen specifically for our scenario, as we do not expect to find more than 30 clusters, but it may be increased if large datasets with potentially numerous clusters are processed. Another parameter would be the projection used, considering the different ones as generated from the previous stage. In the case of the agglomerative algorithm, it can also be parameterized with the linkage type: Ward or average. Once this stage is executed, the end result is essentially a set of predictions, each one representing the output of using a parameterized clustering algorithm from all the possible combinations (e.g., KMeans with k = 2 and using a reduced set of dimensions from a linear Kernel PCA).

3.2.8. Evaluation

The predictions of all the different clustering algorithms can now be evaluated using several functions. Such evaluation is needed to actually carry out the search for the most promising result, considering also that many predictions may be inadequate due to using non-optimal parameters, specially the number of clusters. Even though the pipeline relies on unsupervised ML, it is still possible to use three evaluation metrics, which are also implemented in ScikitLearn, to assess the cohesion of the clusters and the degree to which their samples can be grouped together [50]:

- Silhouette (Sil) [51], a common metric that assesses the consistency of the clusters, measuring the cohesion and the separation of the data points within each cluster and to neighbouring ones. The ideal value is 1, and the worst scenario is −1. Generally, those values above 0.50 can be considered indicative of a possibly existing meaningful clustering, although for our pipeline we opted to use a configurable value of 0.30 as the minimum to take into account a predictor.
- Davies–Bouldin (DB) [52], a metric that measures the distance between clusters and assesses their separation. The ideal value is 0 in this case. We empirically considered a maximum value of 2 for this function, as most predictors with higher values also score badly on other evaluation functions and generally underperform.
- Calinski–Harabasz (CB) [53], a metric that also measures dispersion, both between clusters and inside the clusters themselves, with higher values as better ones. In this case no restriction was applied.

Additionally, other three metrics of the prediction related to the number of clusters and their size have been taken into account for evaluation purposes: the number of clusters (k), and the cluster histogram kurtosis ($kurt$) and skewness ($skew$). These last three metrics can be directly implemented with functions available in Pandas.

Unfortunately, even if these six metrics are useful to measure the fitness of the predictions, they still pose a challenge if we want to use them to order and compare the different predictions. To solve this, it would be ideal to have a single metric that allowed to rank the predictors, which motivated the definition and use of a heuristic that combines the six metrics (see Eq. (1)). For the three evaluation metrics, a scaling is performed in order to map their values to the range [0–1], so they are referred to as $sSil$, sDB and sCB, respectively. Moreover, the use of this heuristic is backed by the fact that none of the evaluation metrics by themselves can provide good predictors for a wide range of different experiments. For example, if we only take the Silhouette metric, it tends to favour heavily unbalanced predictions with very few clusters that in turn either encompass most of the jobs, or only have a few outlier jobs. On the other hand, the Calinski–Harabasz metric heavily promotes predictors that have many clusters. The heuristic ultimately aims to combine all the evaluation information and get those predictors that generally have good cohesion and separation (Sil, DB and CB), while also being balanced in the number of clusters predicted (k) as well as in the size of such clusters ($kurt$ and $skew$).

$$heuristic = (sSil - sDB + sCB) \times \frac{1}{k} \times \frac{1}{abs\,(kurt \times skew)} \tag{1}$$

By applying this heuristic we sort the predictors, so that the best candidates appear at the top. Therefore, their clustering results can be visually evaluated or, if necessary, manually inspected by looking at the job resource plots for each cluster. However, note that some predictors may have the exact same clustering result, which in turn causes them to have the same heuristic value and be grouped together. Taking this into account, the ordered predictors are then processed again to create ranks and remove the spurious ordering between these same-result predictors. This rank position will be the one actually used for the evaluation of the experiments presented in Section 4.

3.2.9. Visualization

Regardless of the heuristic and its best effort to look for the most suitable results, it is still advisable to perform a human assessment on the reports generated by the pipeline, combining their evaluation metrics and several visualization plots, both the resource and the clustering ones. As proven by previous works [54], visualization frequently allows to quickly detect at a glance if a clustering is promising or not, if there are clear outliers that may be anomalies, or if there are easily differentiable groups. In this work, several plotting methods have been used both for the projections, which may be interesting to be evaluated as the input for the clustering, and for the predictions, as the actual result of such clustering.

For the projections, it is possible to use scatter plots and Andrews curves, as they can be applied to the coordinate values of each job. However, there is a limitation in the number of coordinates supported for the scatter plot, with up to three (i.e., 3D scatter plots). In this work, only two-dimensional scatter plots are considered, which use the first two components of the dimension reduction. Fortunately, Andrews curves are able to overcome this issue thanks to their design based on Fourier series, which extends the limitation much further without any theoretical boundary. Nevertheless, we limit such plots to the first five dimensions, considering that any further dimension adds little value in a vast majority of cases and renders the plot visually challenging to be interpreted. In addition, heatmaps can be created for every dimension reduction transformer, where the degree of variation across the jobs can be evaluated for every component. This allows to visually assess the number of components required to capture a great amount of variation. Furthermore, the projections plots can also be used as a hint of the minimum number of clusters required to separate the jobs, previous to any clustering operation.

For the predictors, both scatter plots and Andrews curves can also be used, but in this case with labels according to the cluster, which makes it easy to visually check for the cohesion of the clustering. Moreover, dendrograms can be used to show the hierarchical relationship between jobs and clusters, as well as the relative distance that separates them.

Finally, it is possible to see the actual resource time series plots of the jobs, for any of the metrics and for any predictor result, as well as for outlier jobs. Examples of all these visualization plots will be used for the experiments exposed in Section 4 to enhance the clustering assessment and to complement the evaluation metrics.

Table 3
Computational and memory complexity of the pipeline stages and parallelization approaches.

Stage	CPU	Memory	Parallelization
Feature Extraction	O(j × m × mlen × f) [high]	O(j × m × mlen × f) [high]	Spark and Python
Outlier Detection	O(j × f) [low]	O(j × f) [medium]	None
Dimension Reduction	O(j × f × r) [medium]	O(j × f × r × R) [medium]	Python
Clustering	O(j × r × c) [low]	O(j × f × r × R × c) [medium]	Python
Evaluation	O(j × f × c × e) [medium]	O(j × f × r × R × c) [medium]	None

Legend: j → #jobs | m → #metrics | mlen → metrics length | f → #feature extraction functions | r → #dimension reduction models | R → #reduced dimensions generated per model | c → #clustering models | e → #evaluation functions.

3.3. Computational complexity and parallelization

It is worth noting that the stages previously described are completely decoupled from the data to be processed. This means that they have been developed to be independent of the data size or properties, only considering that such data are essentially time series. Nevertheless, the data volume is key to the performance of the pipeline architecture as the whole process has to be executed efficiently or, at least, in a scalable manner, specially if the aim is to cluster jobs quickly or in short time windows.

Although not a part of the actual pipeline, it is interesting to note that the underlying *Resource Monitoring* process, as well as the *Data Collection* and the *Preprocessing* stages, are inherently scalable. For the latter two stages, because the jobs are independent, their information can be retrieved in parallel from the scheduler database and thus a simple approach of process-based parallelism can be used. And when it comes to the *Resource Monitoring* process, we can count on its scalability because ultimately it is implemented by a collection of distributed monitoring agents deployed on the computing nodes, which in turn have low overhead. In addition, because agents are stateless, as they send all the data as soon as possible to an external database, it is possible to scale this stage if needed as long as the database is capable of ingesting the data. Such database, HBase in our case, has proven to be highly scalable by relying on distributing the data across several instances following a Big Data architecture.

Regarding the computational complexity of the pipeline stages, it is described in Table 3. The variables shown are described as follows: (1) 'j' is the number of jobs, which can be greatly variable, from tens to hundreds of jobs; (2) 'm' is the number of resource metrics, which is usually low due to the limited amount of 'useful' metrics; (3) 'mlen' is the length of the resource metrics, which is also greatly variable depending on the runtime of the user jobs, although it can be modulated through downsampling; (4) 'f' is the number of feature extraction functions, which although variable can be very high if these functions are parameterized with multiple value combinations, some of them also being of high complexity; (5) 'r' is the number of dimension reduction models, usually a low number and of low to medium complexity; (6) 'R' is the number of generated dimensions for each model, usually a low number (e.g., 100) although it depends on the data variability; (7) 'c' is the number of clustering models, which is usually low and of low complexity (i.e., small search space if a dimension reduction is used); and (8) 'e' is the number of evaluation functions, usually a fixed number of functions of low to medium complexity.

Using these variables, the complexity of the CPU and memory requirements for each stage can be analysed, as they both represent the amount of resources needed to run the pipeline. Although CPU usually determines the runtime and is regarded as the most important resource, memory can also be crucial to avoid execution failures caused by out of memory errors. On the one hand, according to Table 3, the highest complexity stage in terms of CPU is *Feature Extraction*, as it depends on many variables with potential high values. Other stages like *Dimension Reduction* and *Evaluation* have a medium complexity, considering that they both depend on the number of features, or low like *Clustering*, which mainly depends on a small set of dimensions as generated by *Dimension Reduction*. The *Outlier Detection* stage also depends on the number of jobs and features, but its complexity is on par or even lower than the *Clustering* stage due to its behaviour based on solely distance calculation or job and feature sampling to construct the detectors. On the other hand, the stage with most memory requirements is also *Feature Extraction*, as it must store all the job metrics on a DataFrame in memory, as well as the features computed from them. However, once such metrics have been used (i.e., m × mlen), they are removed. The remaining stages have a medium memory complexity as they store the features extracted, as well as the results of the previous transformations and their own, that is, all the reduced sets of dimensions (i.e., r × R) or the cluster predictions generated by the *Clustering* stage.

Nevertheless, the complexity of these stages, specially those with high complexity, can be tackled by relying on the parallelization capabilities offered by Big Data frameworks. In our scenario, two parallelization techniques have been implemented: (1) a distributed mode where a Big Data cluster is used to run a Spark workload; and (2) a local mode where the Python multiprocessing library is used to distribute the workload across several processes running on the same host, usually pinned to different cores. This local parallelization has been implemented for some stages of the pipeline, serving as a first line of defence for those complexities that are expected to never become an issue. But for those stages with high complexity it may be required to increase the potential for scalability beyond that offered by a single host. This is of particular importance for the *Feature Extraction* stage, as it is the most intensive one for both resources. That is the reason why for this stage we have used a Big Data cluster, and more specifically, Apache Spark [11] coupled with YARN [55]. With this kind of parallelization the amount of resources is not limited to a single host, but rather to a pool of distributed resources from various hosts, which grants massively parallel processing and enables the *Feature Extraction* stage to scale on par with the complexities of CPU and memory.

Specifically to our scenario, in the *Feature Extraction* stage the entire input dataset is split into small chunks of 80 jobs to be processed using Spark, and for each chunk the jobs are then distributed among the Spark workers, one job at a time [56]. In this way the CPU and memory complexity of the worker is lowered from O(j × m × mlen × f) to O(m × mlen × f), while the Spark driver is also not overloaded as it only has to store one chunk at a time.

Finally, some runtime metrics recorded for the execution of the pipeline stages during the experiments are presented in Section 4.4, offering a hint of the effect that the two parallelization techniques have on the experiments depending on the characteristics of the dataset.

4. Experiments

Several experiments have been designed to show the results of applying the pipeline presented in this paper. The input datasets have been extracted from a working environment represented by a real supercomputer between the 1st of January 2019 and the 1st of July 2020, with each dataset containing jobs from either one or several users [57]. More information regarding the characteristics of the datasets will be given in Section 4.4, where such characteristics are used in conjunction with the pipeline runtimes to assess its scalability.

The experiments have been divided into three groups: experiments that use datasets containing jobs from a single user, those with datasets

Table 4
Experimental results for single-user job clustering.

Experiment	#Predictors #Ranks	Projection	Predictor	Sil	DB	CB	Job distribution	Outliers
Single User 1	58 39	Kernel PCA 100 components linear	KMeans (#7)	0.334	1.759	35	128 jobs in 3 clusters 92| 21| 15|	8 outlier jobs, ranked: 0| 12| 13| 29| 39| 41| 56| 100|
Single User 2	61 42	Spark PCA min variance of 90%	Agglomerative (#0) Euclidean distance Ward linkage	0.402	0.912	233	445 jobs in 4 clusters 247| 183| 14| 1|	25 outlier jobs
Single User 3	44 30	Kernel PCA 100 components polynomial degree 3	Agglomerative (#4) Euclidean distance Ward linkage	0.333	1.211	23	127 jobs in 8 clusters 73| 12| 12| 9| 6| 6| 5| 4|	6 outlier jobs, ranked: **0**| 24| 30| **45**| **74**| **100**| (bold → displayed)
		Kernel PCA 100 components polynomial degree 2	KMeans (#3)	0.332	1.179	28	127 jobs in 5 clusters 78| 21| 14| 8| 6|	

containing pairs of these same users, and experiments that use a time window which encompasses many users. For the single-user experiments, outlier-based anomaly detection has also been studied, considering that in such experiments the job variability is lower (i.e., users tend to repeat a few job templates or patterns), thus making the anomalies stand out more. Although the anomalies are analysed, their interpretation or possible connection to erroneous executions is ultimately left to the user or system administrator, and only some guessing can be done as to what caused the job to stand out, mainly by looking at the resource plots.

4.1. Single-user job clustering

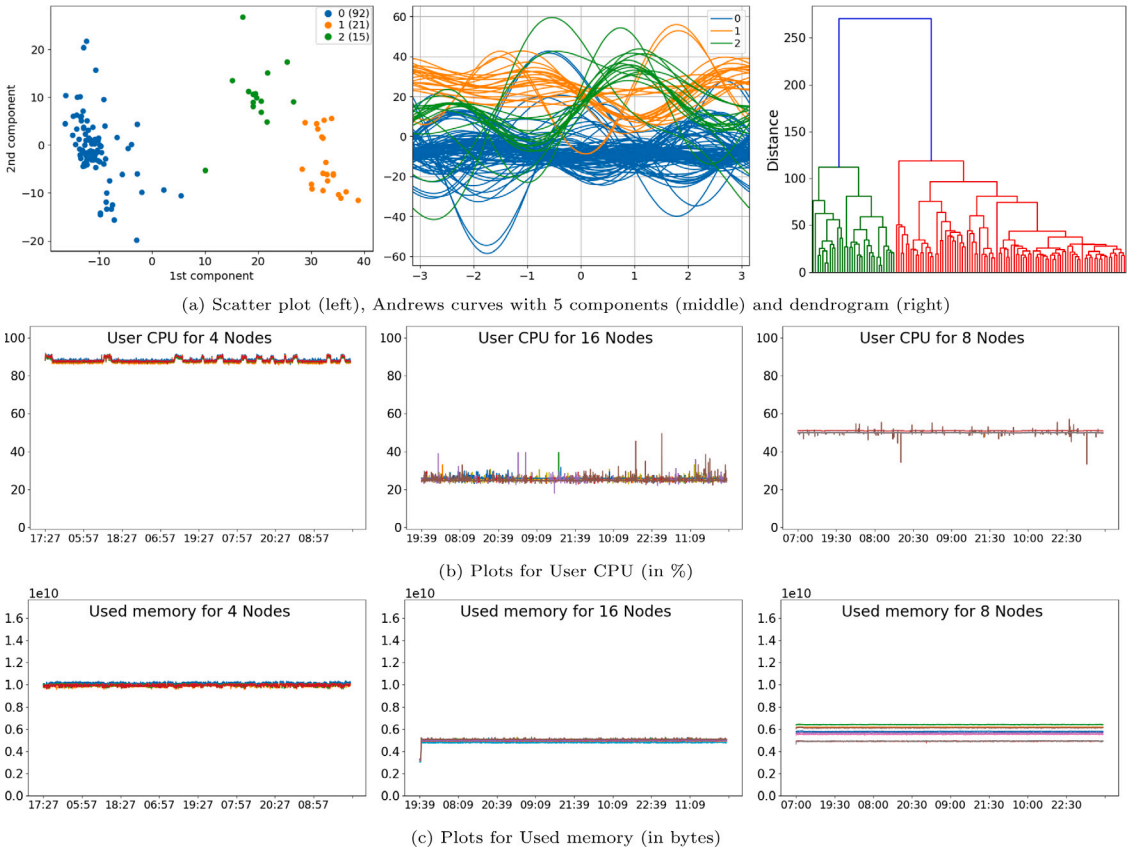
For these experiments, a total of three users (named 1, 2 and 3) have been selected. Their selection was based on having datasets with potentially different characteristics, from low or medium job variability to high variability, as well as different dataset sizes and job runtimes. The datasets have been extracted from a pool of users that have executed a minimum of 100 jobs with a job duration of at least 90 min. For each experiment, several plots are shown containing both the visualization used as validation (top row in the forthcoming figures), and the resource plots for the different clusters (second and third rows). For these latter plots, a different resource is displayed in each row, with each plot line representing a node and the title describing the resource used and the number of nodes. Moreover, a different cluster is shown in every column using a representative job, and in turn the column order is the same as the cluster number from the validation plots (e.g., the resource plots of the second column are from a job of cluster 1). For some experiments, representative outlier jobs will also be displayed by showing their resource plots. In addition, the predictor configuration and its evaluation metrics are described in Table 4.

4.1.1. User 1

For the first user (see Fig. 3), a relatively simple scenario is shown where well-defined clusters can be isolated. The top row of validation plots (see Fig. 3(a)) shows from left to right the scatter plot of the chosen predictor, the Andrews curves for five dimensions and a dendrogram. The scatter plot allows to identify one big cluster, as well as two smaller ones. This can be further backed using the Andrews curves for this same predictor, where two clusters are clearly differentiated mainly by using the first component (i.e., the constant term, the DC component of the curves), or if necessary, the second component (i.e., the first harmonic of the curves). The last validation plot, the dendrogram, also shows that if a hierarchy is used, two clusters are separated early (i.e., great height or distance), one of which (green coloured on the left) could be further split in a balanced fashion into two clusters containing the second and third smaller ones in this case. The second and third rows (Figs. 3(b) and 3(c), respectively) show the resource plots, which can be used to prove how the clustering is meaningful, considering that the dimension reduction phase associated a total of 56% of the variance to features extracted from the User CPU (41%) and Used memory (15%). For this user and dataset, the main difference between the clusters would then be for the User CPU metric, as can be seen in Fig. 3(b), while memory could also be used to a lesser extent. When it comes to the evaluation metrics, as displayed in Table 4, the chosen predictor for this experiment is placed in the eighth rank (#7) by the heuristic out of a total of 39 ranks from 58 predictors that satisfied the minimum requirements previously exposed in Section 3.2.8. Additionally, eight outliers were detected.

4.1.2. User 2

For the second user (Fig. 4), a slightly more complex scenario is depicted as the number of different job types seems to be larger, that is, the job variability is higher. For the validation plots (top row), we can use the scatter plot on the left to see that there are four clusters in total: two big clusters easily differentiated, an additional smaller one, as well as a single-job cluster. The middle and right plots show the Andrews curves for two and five components, respectively, in order to show the reason behind the limitation of the scatter plot. While the scatter plot succeeds in differentiating two groups of clusters, it is not able to fully back the existence of four clusters, rather than two. On the left side of the scatter plot, it is hard to say why the single-job cluster (red coloured) has been isolated, since it is spatially close to its neighbouring and larger cluster on both components. On the right side of the plot, it may be dubious that the small cluster (green coloured) is different enough from its large neighbour one (blue coloured), since it is spatially close if we use the first dimension. This ultimately causes their generated two-dimensional Andrews curves to be similar as well, specially for the single-job cluster. However, when using five components as on the right validation plot, it is easy to see that the added components allow differentiating further, strongly backing the single-job cluster as clearly different from its neighbour one, as well as showing a different trend for the small, green coloured cluster that sets it apart from the large, blue coloured one. Interestingly, there is an additional job that could be differentiated from this large cluster, and that is easily spotted both on the scatter plot (upper right corner) and in the Andrews curves, but that the predictor did not place on a cluster of its own. Even though these 'isolated' jobs have made it through the outlier detection, with these visualization plots in hand, it may be interesting to consider them as outliers and possibly as anomalies. Regarding the plots in the second and third rows (Figs. 4(b) and 4(c)), they allow seeing the differences with the resource metrics using the Cached memory and the User CPU, respectively, which were the second and fourth metrics that captured the most variance according to the *Dimension Reduction* stage. From the Cached memory, it is easy to see that although the first three clusters share the fact of having just one node actively consuming the resource, they show two different patterns, with the first and third clusters being similar, and the second one being different. In order to distinguish these two similar clusters, we can use the User CPU resource, observing now how these clusters have a different behaviour towards the end of the

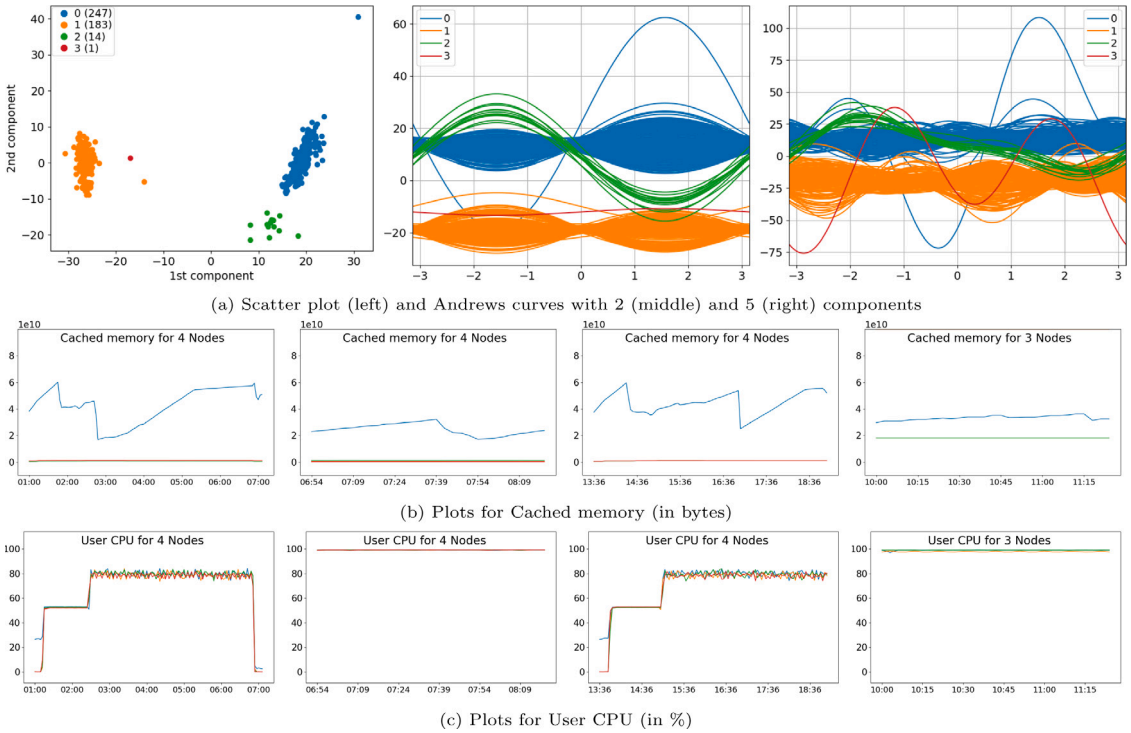


(a) Scatter plot (left), Andrews curves with 5 components (middle) and dendrogram (right)

(b) Plots for User CPU (in %)

(c) Plots for Used memory (in bytes)

Fig. 3. Visualization of results and resource plots for the Single User 1 experiment. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) Scatter plot (left) and Andrews curves with 2 (middle) and 5 (right) components

(b) Plots for Cached memory (in bytes)

(c) Plots for User CPU (in %)

Fig. 4. Visualization of results and resource plots for the Single User 2 experiment. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

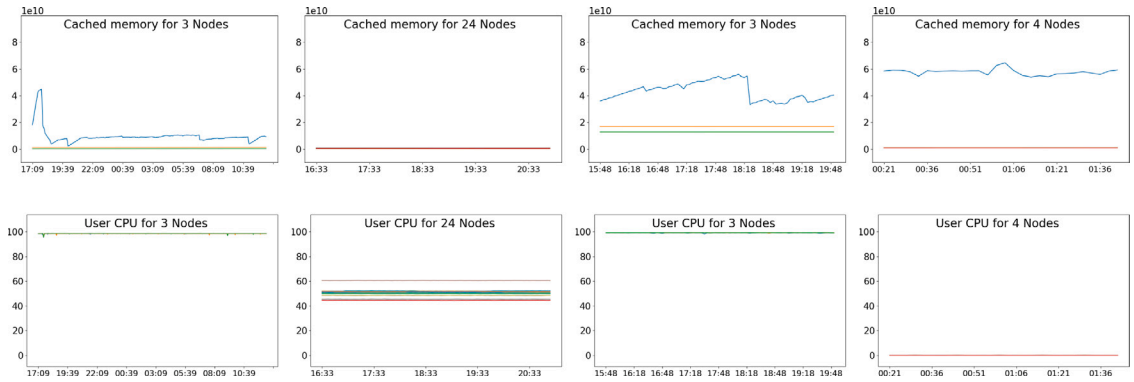


Fig. 5. Single User 2 outliers, showing Cached memory (in bytes) and User CPU (in %) resources.

execution. When it comes to the single-job cluster (far right plots), although its User CPU may be similar to the second cluster, which would explain its initial spatial closeness, its Cached memory clearly shows an anomalous pattern, which backs the previous idea of this job being an actual outlier and an anomaly. For the evaluation metrics (see Table 4), the chosen predictor in this case would be the first one (#0), as suggested by the heuristic out of 61 predictors grouped in 42 ranks.

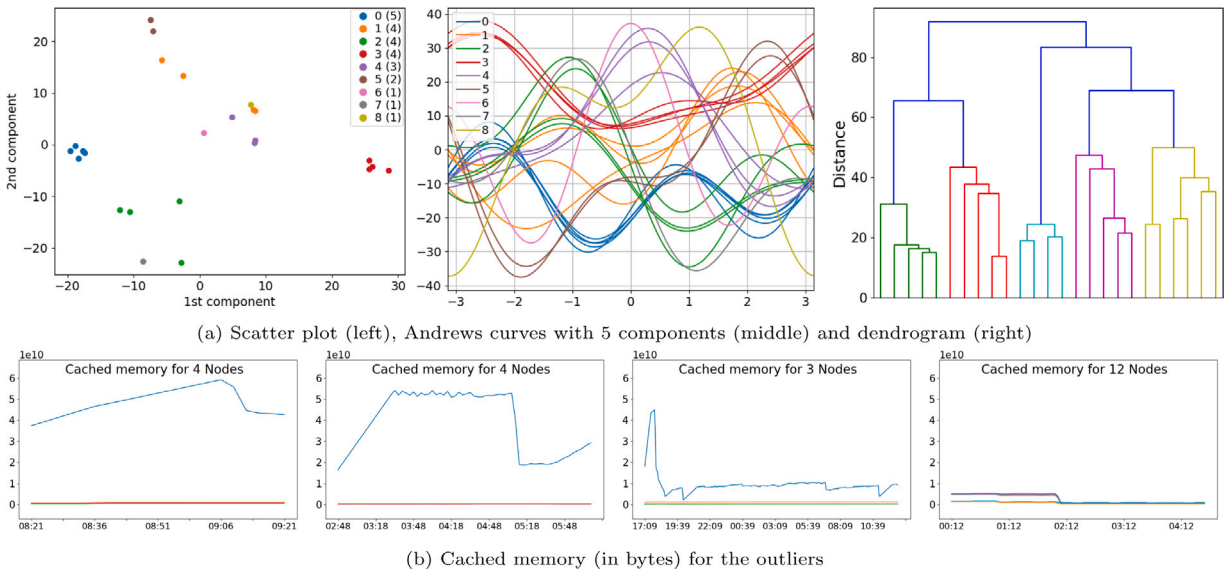
Nevertheless, a total of 25 outliers have been detected for this second user, which is a relatively high number as it represents approximately 5% of the original number of jobs (470). Some of the most relevant outliers according to their rank can be seen in Fig. 5. By using the same resource plots as in Fig. 4, it can be seen that they are radically different from the previous detected clusters on one resource (e.g., first and third columns) or even both (e.g., second and fourth). Considering the high number of outliers for this dataset, it may be interesting to use the pipeline to process such outliers in the search of groups or associations between them. The results of this processing can be seen in Fig. 6. Note that we are applying the same clustering pipeline, but with a low number of jobs (even though they represent a high number of outliers), to a group of jobs that were previously discarded for being too different and abnormal. The combination of these facts explains why an unusually high number of clusters (9) was detected, most of them containing four or fewer jobs. Nevertheless, the validation plots (see Fig. 6(a)) can be used to assess that some of these clusters apparently have a cohesion, which indicates that they are also somehow related, even though they were outliers as a whole. The same Cached memory metric as used earlier can now be seen for 4 out of the first 5 clusters in Fig. 6(b). Although a job belonging to one of these clusters (the third one) has been presented before as a representative outlier (see the first outlier in Fig. 5), the remaining clusters and jobs were previously left out due to a lower score when the outliers were ranked.

Considering now that the outlier detection is optional, we analyse the scenario where the outliers were not taken out of the dataset. The results for such scenario are available in Fig. 7. The most direct effect of not removing the outliers is a shift in the cluster composition. Even though the number of clusters is the same as before, we have now two large clusters and two minimal ones when looking at the validation plots (see Fig. 7(a)). These two large clusters are actually the first three clusters as before with a difference. Previously, two clusters (first and third) would tell apart those jobs according to their User CPU usage near the end of the execution (see Fig. 4(c)), but now they are fused into one cluster. The other two minimal clusters are most probably fully composed by outliers, considering that two previous detected outliers are present (see third and fourth plots in Fig. 7(b)). This scenario points out that the outlier detection and removal phase is not actually essential, even though it can be advantageous to take out those jobs from the dataset so not to misguide the feature extraction functions and the clustering models. Even when the outliers were not removed, the larger and most important clusters were detected. Nevertheless, if the objective and use case of applying clustering is either to detect such outliers, or to look for smaller groups of jobs that deviate from such larger clusters, then removing the outliers is crucial.

Finally, it is worth commenting for this experiment overall how these groups of anomalous jobs, as seen in Fig. 6 and once analysed, can be useful to detect recurring failing jobs. On the one hand, if the similar anomalous jobs are always attached to a specific user, these jobs could be associated with erroneous job configurations. On the other hand, if present across different users, they may be linked to a failing infrastructure component (e.g., wrong configuration of a node, failing piece of hardware). Nevertheless, in both scenarios it would be interesting to notify either the user or the system administrator, respectively, to take corrective actions.

4.1.3. User 3

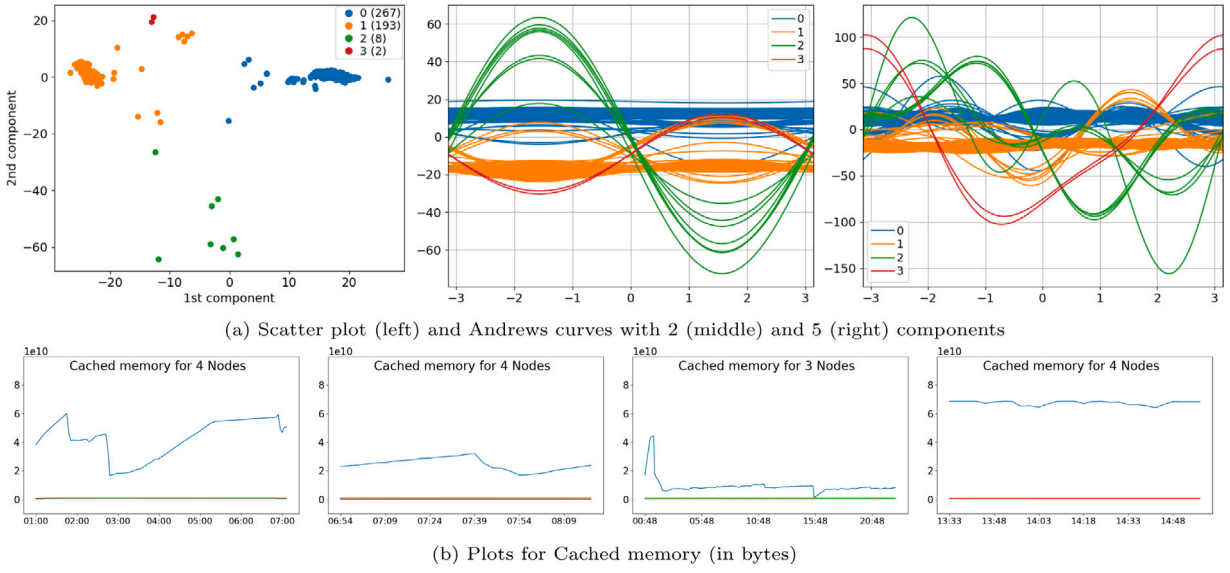
The third user represents a scenario where there is a high variability across the jobs (see Fig. 8). In this case, two predictors (see Table 4) are shown in the top row for the validation plots (Fig. 8(a)). According to the scatter plots, it can be seen that between five and eight clusters may be present, although it is much more difficult to prove this statement without using further data, as several clusters are either too close together or may even appear to overlap. This is mainly due to the 2D limitation of the scatter plots. However, such overlapping clusters can be decoupled to some extent when analysing the Andrews curves on the right plot by using more than two components. Nevertheless, the two scatter plots are interesting to be compared if we take into consideration that, although both use a polynomial dimension reduction, the first uses a third degree one, while the second uses a second degree. This difference can also be seen in the fact that the clusters tend to be arranged along lines as the number of degrees increases. Regarding the resource plots (Figs. 8(b) and 8(c)), they show the first four out of the five possible clusters based on the clustering results from the second predictor (#3). In this case, User CPU and Used memory are shown, which were the metrics that backed the features with the most variance captured according to the *Dimension Reduction* (43.5%), albeit that information is given by a linear reduction model. As can be seen, the jobs exhibit clearly different patterns for both resources. The difference between the predictors, as specified in Table 4, besides the different dimension reduction used, is that they also use different clustering models, being nonetheless ranked #3 and #4 by the heuristic from 30 possible ranks. When it comes to outliers, a total of six were detected, with four of them displayed in Fig. 9, including the three with the highest scores (see Table 4). As can be appreciated, the first three outliers correspond to jobs that followed similar patterns to the clusters in Fig. 8, but that at some point suffered some kind of error that altered the normal execution, while the fourth is probably linked to a failed job from the start of the execution. This information can be useful to detect job executions whose pattern deviates significantly from the expected one. However, unlike User 2, in this case the anomalies were less



(a) Scatter plot (left), Andrews curves with 5 components (middle) and dendrogram (right)

(b) Cached memory (in bytes) for the outliers

Fig. 6. Single User 2 outliers clustered.



(a) Scatter plot (left) and Andrews curves with 2 (middle) and 5 (right) components

(b) Plots for Cached memory (in bytes)

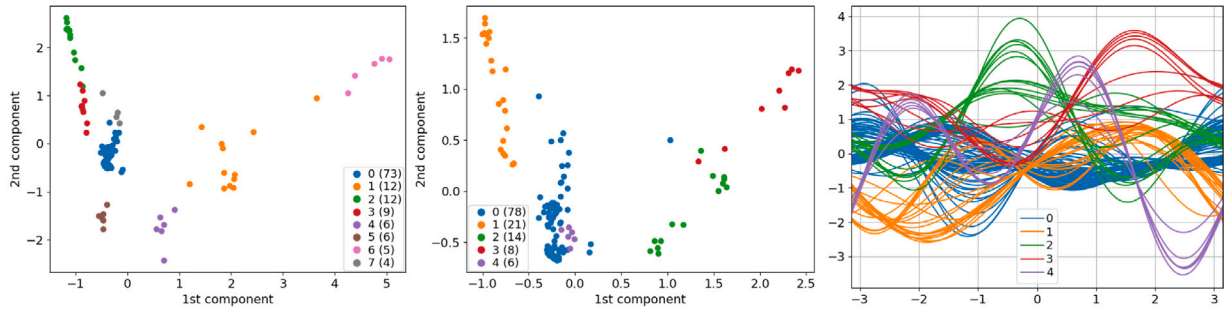
Fig. 7. Single User 2 experiment keeping the outliers.

numerous and more specific. This fact discards any need for clustering and points out that these anomalies are less likely to be linked to a recurring failure, and that they have to be manually inspected for more information. Nevertheless, they can still suggest an isolated software failure, or a piece of hardware that started to fail recently, as maybe not enough anomalies have been accumulated yet.

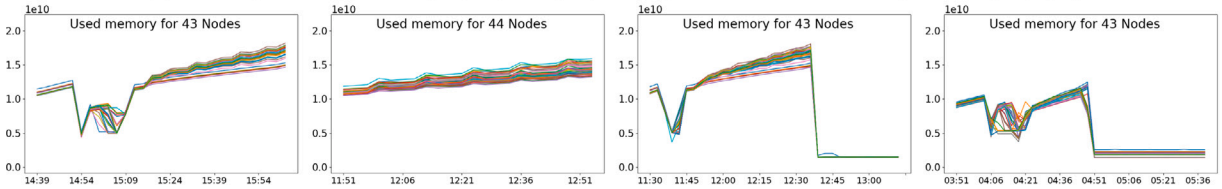
4.1.4. Heatmaps analysis

In order to further understand the reasons behind the predictions presented in the previous single-user experiments, the heatmaps shown in Fig. 10 can be analysed, as they ultimately display the data used by the clustering algorithms. Nevertheless, note that each row in the heatmap represents a job previously to any grouping, and thus the values are not ordered. Furthermore, only the first five components are displayed considering that adding further ones hinders visualization. For User 1, there is a relatively medium variability overall, requiring in this case 38 components to capture at least 90% of variance, a piece of information we can extract from the *Dimension Reduction* stage as previously explained in Section 3.2.6. If we look at the first component, there could be at least two dominating big clusters and possibly other minor ones. However, if we look at the next components, we can clearly see some radically different values that could belong to undetected outliers jobs, or slightly different jobs inside the large clusters. Regarding User 2, we can also see clearly the two big clusters using the first two components, as previously seen in the scatter plots, and some different values in the second component, which may explain the third cluster. Unfortunately, as opposed to the Andrews curves, it is very difficult to detect the outliers previously mentioned using the heatmaps, both the single-job cluster and the undetected one, as they are buried among the other numerous jobs. In this case, the projection used needed 43 components to correctly capture the variability. For User 3, we can observe a relatively high variability in the fact that up to 50 components were usually required to capture enough variance, and that visually all the components, even the first one, contain several different grades of values for several jobs.

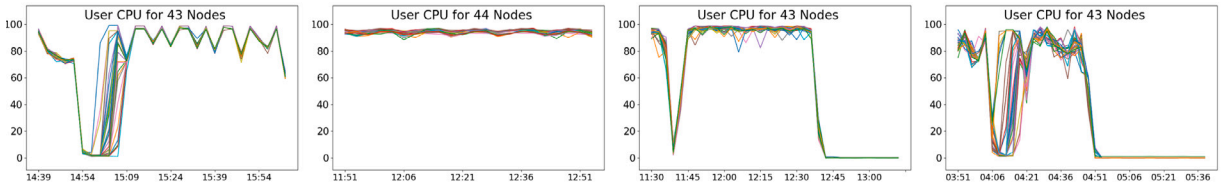
Finally, it is worth commenting that the experiments shown in this section present two use cases that can be useful in our HPC scenario to system administrators. First, the clustering allows extracting



(a) Scatter plots for predictor #4 (left) and #3 (middle), and Andrews curves for predictor #3 (right)



(b) Plots for Used memory (in bytes)



(c) Plots for User CPU (in %)

Fig. 8. Visualization of results and resource plots (first 4 out of 5 clusters, predictor #3) for the Single User 3 experiment.

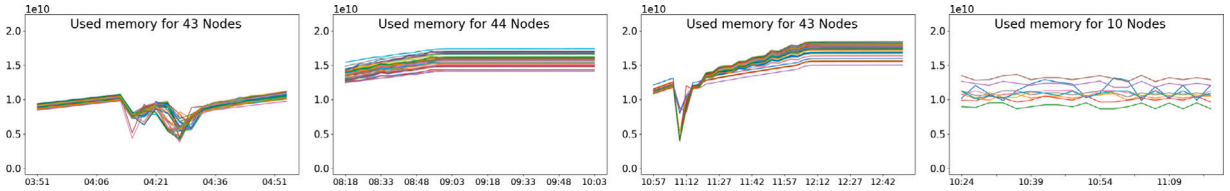
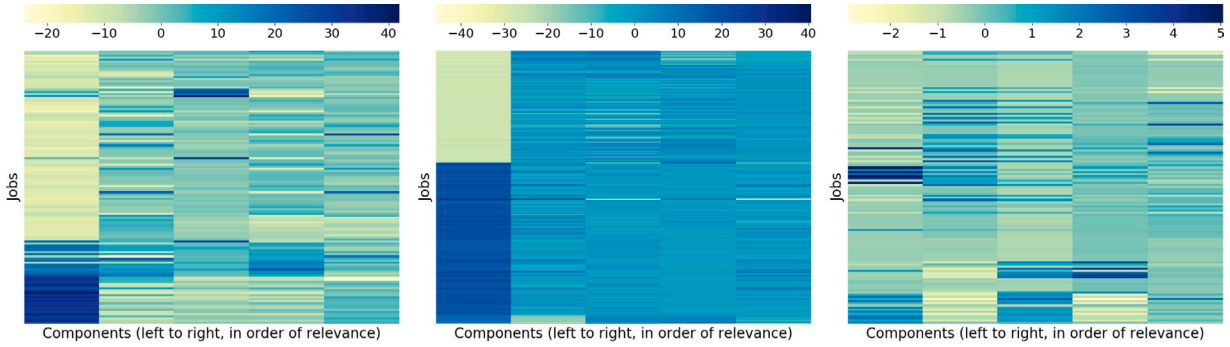


Fig. 9. Single User 3 outliers, showing Used memory (in bytes) resource.



(a) User 1 – Kernel PCA linear (b) User 2 – Spark PCA (c) User 3 – Kernel PCA polyn· degree 3

Fig. 10. Heatmaps for the single-user experiments.

information from job datasets, like the type of jobs a user typically executes, mainly their resource patterns and variability both between the jobs themselves and compared to other job types (see Figs. 3, 4 and 8). Second, and derived from the first use case, we can also extract the outliers from a dataset, which could be labelled as anomalies after human inspection. This would reveal if a job execution has suffered from some kind of error and, using its resource pattern, try to diagnose it (see Figs. 5 and 9). Furthermore, these anomalies can be clustered to search for repeated execution failures or similar kinds of anomalies (see Fig. 6).

4.2. Multiple-user job clustering

For these experiments, the previous users have been combined by pairs. This allows us to see scenarios having some idea of what the result should resemble. For example, a certain number of clusters

Table 5
Experimental results for multiple-user job clustering.

Experiment	#Predictors #Ranks	Projection	Predictor	Sil	DB	CB	Job distribution
User 1+2	141 119	Kernel PCA 100 components polynomial degree 2	KMeans (#0)	0.459	0.881	235	583 jobs in 5 clusters for 2 users User 1: | 7| 87| 33| | User 2: 264| 191| | | 1| 23 outliers found
User 2+3	142 123	Spark PCA min variance of 90%	Agglomerative (#16) Euclidean distance Ward linkage	0.447	1.385	239	574 jobs in 5 clusters for 2 users User 2: 263| 186| 1| | | User 3: | 8| 61| 37| 18| 29 outliers found
User 1+3	44 30	Kernel PCA 100 components polynomial degree 2	Agglomerative (#1) Euclidean distance Ward linkage	0.331	1.067	65	259 jobs in 5 clusters for 2 users User 1: 8| 84| 33| | 1| User 3: 110| | | 23| | 10 outliers found

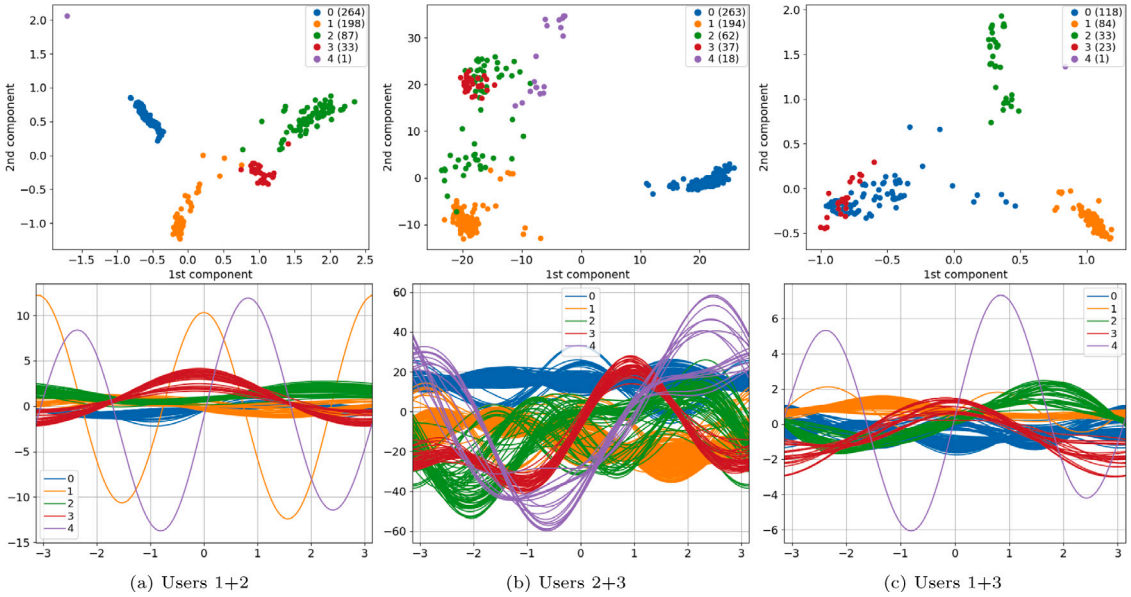


Fig. 11. Visualization results for multiple-user job clustering.

could be estimated, taking into account the results from the previous single-user experiments. Ideally, as long as the underlying user jobs are different enough, a robust prediction should keep the clusters from both users separated. The results for these experiments are available in Table 5, and Fig. 11 shows the three combinations of the users. Each combination is displayed in a column, being the top graph the scatter plot of such combination, and the bottom one the Andrews curves.

The first column (see Fig. 11(a)) combines the experiments from Figs. 3 (User 1) and 4 (User 2), resulting in five clusters extracted by the predictor (the first one out of 141 predictors across 119 ranks). Similarly to single-user experiments, this can be seen using the scatter plot up to a certain degree (see top graph), where the clusters are clearly differentiated, even keeping the single-job cluster from User 2. The Andrews curves (bottom graph) further back this analysis. Additionally, the previously undetected outlier for User 2, which the scatter plot is still unable to detect, is also present in the curves plots. Moreover, the evaluation data shown in the last column of Table 5 (Job distribution) allows assessing if the clustering has somehow taken into account the fact that the jobs present in the dataset come from two users. In this case, the five clusters have mostly maintained the results from the previous experiments for the users individually (two large clusters for each user), only mixing a few jobs from both users or moving them to another cluster, which is not necessarily a sign of a bad clustering as some jobs in particular may indeed be similar. It has also to be taken into account that different outliers could have now been dropped if compared with the individual experiments.

The second column (see Fig. 11(b)) combines the experiments from Figs. 4 (User 2) and 8 (User 3). In this case, the predictor extracts five clusters which, as before, are more or less related to the previous results from the users individually if we take into account the large, or medium-sized clusters that are close. Unfortunately, by looking at the scatter plots we can see how the points begin to overlap substantially. Looking at the evaluation data in Table 5, we have to go down to the predictor ranked #16 when the results are visually inspected, which proves that the heuristic may still fail in some cases to highlight potentially interesting results. In addition, the single-job cluster and the undetected outlier for User 2 do not seem to be present, neither on the scatter plot nor on the Andrews curves. This could mean that they are not so different now when compared to other jobs, or they have been properly detected and dropped as outliers.

The third column (see Fig. 11(c)) mixes the experiments from Figs. 3 (User 1) and 8 (User 3). The predictor shown in this case was the second one (#1) selected by the heuristic out of 30 ranks. The scatter plot seems to properly separate some clusters, but two of them are severely overlapped, although they can still be separated using the Andrews curves. This effect is mainly due to the polynomial dimension reduction used, which may cause the effect of grouping along lines, as previously stated. Interestingly, a single-job cluster is also produced in this experiment, which may be due to an outlier that was not dropped, or to a job that is now radically different as the result of combining the two datasets.

Table 6
Experimental results for time-window job clustering.

Experiment	#Predictors	Projection	Predictor	Sil	DB	CB	Job distribution
Time window 0	97 73	Spark PCA min variance of 90%	Agglomerative (#1) Euclidean distance Ward linkage	0.311	1.347	60	197 jobs in 6 clusters for 7 users 9 outliers found
Time window 1	59 46	Kernel PCA 100 components linear	KMeans (#3)	0.350	1.121	41	141 jobs in 6 clusters for 6 users 9 outliers found
Time window 2	92 63	Kernel PCA 100 components polynomial degree 2	KMeans (#2)	0.372	1.340	68	163 jobs in 4 clusters for 5 users 13 outliers found

This scenario with user pairs can serve to prove or assess the degree of similarity between the jobs that two users execute, by using the visualization as well as the job distribution (see last column in Table 5). Even though they were not shown for brevity and to avoid repetition, it would be possible to compare the clusters using the usage patterns from the resource plots. The results could point at two users whose jobs are similar or totally opposite, and if the users have different job types it may even show if any of those is shared by both users. In our case, the three users have been fairly well separated and thus it can be asserted that they have different job patterns. Although this may be extensible to combinations of three or more users, complexity increases with each added user. Nevertheless, this use case may be interesting if extended to many users to extract job patterns that are heavily used. Once detected and studied, these patterns can be the basis for guiding several decision-making processes. For example, in the short to medium term, these patterns can be useful to improve the configuration of the job scheduler (Slurm in our case), or to create different resource constraint policies for job executions (e.g., Slurm partitions) to accommodate such patterns, all in order to increase hardware utilization and efficiency. Furthermore, in the medium to long term, these patterns can serve to guide the upgrade or acquisition of an HPC infrastructure by knowing how the resources were used in the past by users.

4.3. Time-window job clustering

For the last experiments, several time windows of a week within the first six months of 2020 were defined. These windows may comprise many users, but only those that submitted between 10 and 100 jobs were taken into account to avoid numerous outliers due to users that sporadically run a few jobs, or users that may run a really large amount of jobs, usually with little to no variation among executions. Nevertheless, the remaining jobs still have to abide by the minimum requirements (i.e., job duration and minimum number of nodes) as specified in Section 3.2.1. The results for these time-window experiments, including the evaluation metrics, are detailed in Table 6, and the visualization plots are displayed in Fig. 12. The plots are organized in columns with, from top to bottom, scatter plots, Andrews curves and heatmaps. For these experiments the number of users, and thus the expected job variability, begins to increase substantially when compared to the previous experiments, going as far as seven users.

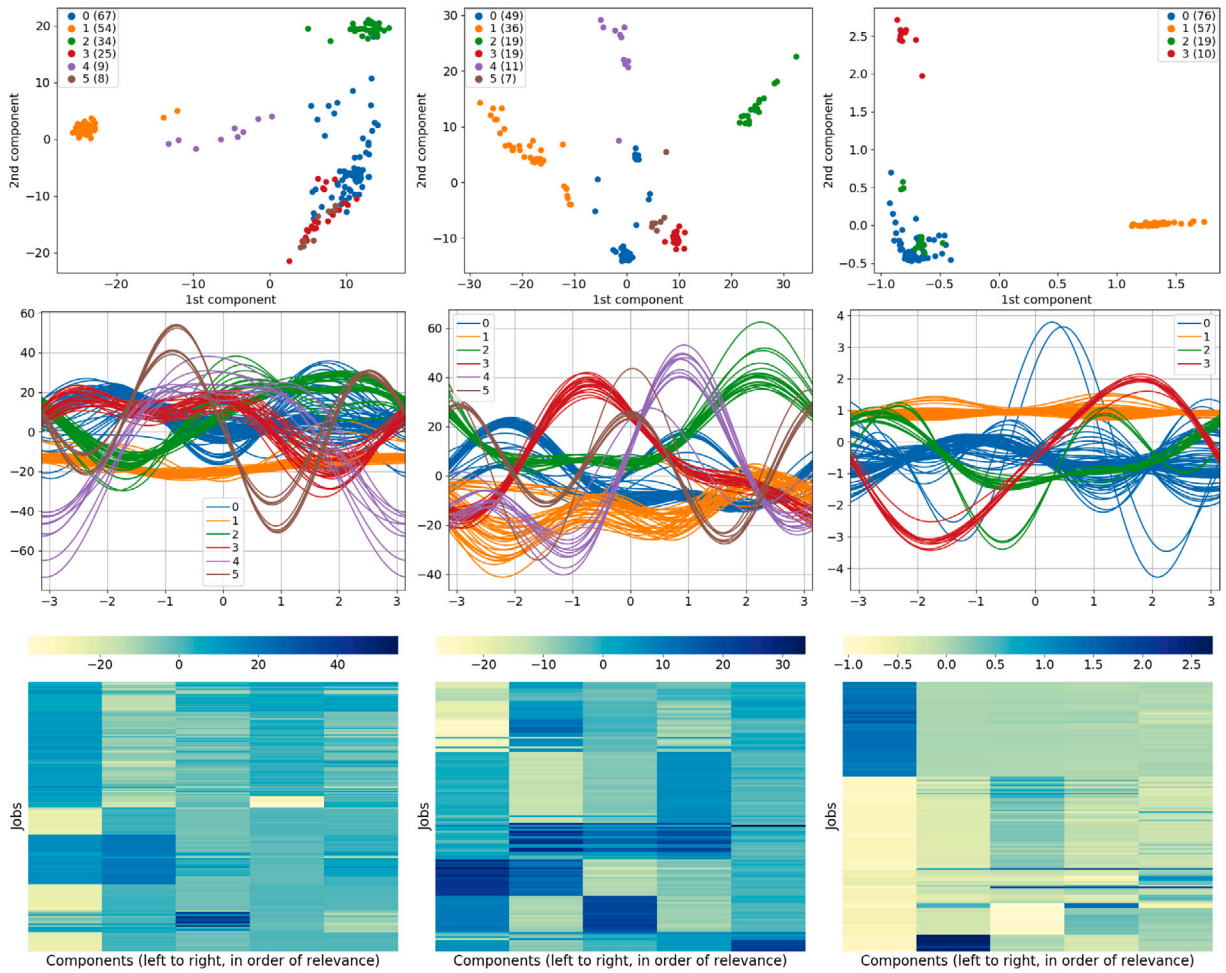
For the first time window (Fig. 12(a)), the result of an Agglomerative clustering after a dimension reduction using a Spark PCA is displayed, ranked second (#1) out of 73 (see fourth column in Table 6). Both with the scatter plot and the Andrews curves, large groups can be isolated and identified. However, while in the scatter plot several large clusters are fused together (clusters 0, 3 and 5), they are easily separated using the Andrews curves. Interestingly, the heatmap also shows that the number of components needed starts to grow, because variability significantly increases by adding more users, requiring up to 41 components to capture enough variance. In this case, all the five components show variability across the values. The second time window (Fig. 12(b)) uses a KMeans prediction coupled with a linear Kernel PCA and is ranked #3 out of 46. The scatter plot allows to identify several cohesive clusters, which in this case are mostly associated with users. However, the additional components of Andrews curves allow to better explain the assignment of those points, hinting that cluster 0 (blue coloured) could possibly be divided into two. When analysing the heatmap, all the components show variability, which is backed by the fact that 34 components are needed to achieve the 90% of variance captured. Finally, a KMeans clustering is performed for the third time window (Fig. 12(c)) using a dimension reduction with a second degree polynomial Kernel PCA, and ranked #2 out of 63. In this case, the scatter plot separates the jobs into four clusters, two of which overlap substantially. This is resolved with the Andrews curves, although this plot reveals that cluster 2 (green coloured) could actually contain two different job types, and also the presence of two outliers inside cluster 0 (blue coloured). The heatmap, in contrast to the previous two ones, comes from a polynomial dimension reduction, which should implicitly cause the values to be less ambiguous and more polarized. However, the variability present in the experiments still forces using several components. Interestingly, due to the smaller dataset size, the two undetected outliers could be spotted in the heatmap as two jobs whose component values are high, but only from the second component onwards. This could be the reason why they have been clustered inside a large cluster and do not appear in the scatter plot, where the first two components are used, but they are highly noticeable in the Andrews curves.

These last experiments may prove useful for a general analysis of the jobs executed in a given time frame. If only the 'active' users are taken into account, that is, those users that execute a minimum number of jobs within such time frame, the results could show a rough picture of the most common job types. This in turn could be interesting to later analyse the resource requirements of such types, which is interesting to prioritize resource provisioning in the future, as already explained at the end of Section 4.2.

4.4. Infrastructure used and runtime analysis

Considering that the pipeline implementation combines two parallelization techniques to overcome scalability issues, a distributed Spark-based parallelization (using a Big Data cluster) and a process-based parallelization (using a local host), it is interesting to analyse the runtimes alongside the complexity of the experiments (previously commented in Section 3.3). This study is even more important taking into account the potential for processing large datasets, whether in terms of a large number of jobs, fewer but long-running ones, or massively parallel jobs (i.e., using a very high number of nodes).

The *Feature Extraction* stage uses the first parallelization technique, considering that it is by far the most compute-intensive stage, while the *Dimension Reduction* and *Clustering* stages use the second one. The *Evaluation* and the *Outlier Detection* stages are not parallelized. Regarding the Spark parallelization, it is modulated by the number of Spark executors used, while the local process-based parallelization is configured with the number of processes that are spawned. The hardware specifications of the infrastructure used in the experiments are detailed in Table 7, both for the nodes of the Big Data cluster that runs the Spark workloads, as well as for the local, single host.



(a) Time window 0 (b) Time window 1 (c) Time window 2

Fig. 12. Visualization results and heatmaps for time-window job clustering. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 8 exposes the parameters of the experiments that determine their complexity (first four columns), as well as the runtimes for different parallelization configurations used for their execution (the remaining columns). The parameters that play a significant role in the total runtime of each experiment are the number of jobs and nodes (second column), and the amount of actual data to be processed, which can be measured as the added up length of all the job runtimes (third column), or also as the total number of data points to be processed (fourth column). Regarding parallelization, three different configurations have been evaluated. For each experiment, the runtime results for each pipeline stage are shown in consecutive rows: 20 Spark workers and 1 process (no parallelization) in the first row, doubled to 40 workers and 2 processes in the second row, and doubled again to 80 workers and 4 processes in the third row.

The impact that the variability has across the datasets can first be analysed by looking at the single-user experiments in Table 8. Even though User 1 is the one with the longest total job runtime, User 2 has the highest number of jobs, while User 3 has the most number of total nodes (i.e., raw number of time series). All these parameters can have some impact when processing the datasets, for example, it is not the same to process a dataset with long but 'low-varying' time series (case of User 1), than with shorter but more complex time series (case of Users 2 and 3). Regarding the parameters of the multiple-user experiments, in the end they are derived from the single-user ones. These parameters are harder to interpret for the time-window experiments if datasets as a whole are taken into account, as the jobs included can have widely different characteristics, specially when many users are present. Nevertheless, it is interesting to note how the parameters for these time-window datasets show little variation among them. This can be explained by considering that the jobs executed on a supercomputer tend to be a few types of periodically repeated experiments, with some sporadic executions of specific jobs by some less active users.

Secondly, the runtimes of the experiments for each stage can also be analysed in Table 8, where it can be seen that the *Feature Extraction* stage is the most time-consuming one. This is even more evident when taking into account that the resources used in this stage are considerably larger: 20 Spark workers for every local process spawned to run the other stages. If we compare the three single-user experiments, we can now experimentally observe that the dataset from User 2 takes longer to process than the one of User 1, even when it is technically smaller, something that may be due to the intrinsic complexity and variability of the time series as previously explained. When analysing how the runtimes correlate to the resources used and the parallel configuration, it is clear that runtimes are reduced when increasing the resources. In some cases, the runtime is close to being halved, such as for User 2, when the Spark workers are doubled from 20 to 40. However, the gain is lower when the resources are doubled again. It should be taken into account that the runtimes start to get significantly smaller when the maximum parallelism is employed (80 workers), and thus it also gets increasingly difficult to achieve a noticeable improvement. Performance gains are also observed when analysing the runtimes of

Table 7
Hardware specifications of the infrastructure used.

	Big Data cluster — 14x nodes	Local host
CPU model	2x Intel Xeon E5-2620 v3 Haswell-EP	AMD Ryzen 9 3900X
CPU speed	2.4 GHz (3.2 GHz in Turbo mode)	3.8 Ghz (4.6 GHz in Turbo mode)
# Cores/Threads	12/24	12/24
Memory	64 GiB DDR4	64 GiB DDR4
Disks	1x 480 GiB SSD SATA 2.5″ (master) 12x 2 TB NL SATA 3.5″ (worker)	1 SSD SATA 2.5″
Network	1 × 10 Gbps + 2 × 1 Gbps	N/A
Operating System	CentOS Linux release 7.2.1511	Ubuntu 20.04
Spark distribution	2.4.0-cdh6.1.1	N/A
Executors per node	8	
Memory per executor	2 GiB	
Cores per executor	1	

Table 8
Dataset characteristics and runtimes for all the experiments (1st row: 20 workers-1 process, 2nd row: 40 workers-2 processes, 3rd row: 80 workers-4 processes).

Experiment	#Jobs #Nodes	Total job runtime (hours)	#Time series points	*Feature Extraction* (seconds)	*Dimension Reduction* (seconds)	*Clustering* (seconds)	*Evaluation* (seconds)	Pipeline runtime (seconds)
Single User 1	136 920	12,748	256,000	240 142 105	17 18 18	6 5 4	6 6 6	269 171 133
Single User 2	470 1,915	1,844	39,000	312 172 108	17 17 18	9 8 6	12 11 11	350 208 143
Single User 3	133 5,709	135	3,000	194 112 71	14 14 14	5 4 3	6 6 6	219 136 94
User 1+2	606 2,835	14,592	295,000	532 297 180	20 19 19	10 7 6	16 16 16	578 339 221
User 2+3	603 7,624	1,979	43,000	473 261 150	19 19 19	11 8 6	16 15 15	519 303 190
User 1+3	269 6,629	12,883	260,000	419 239 172	16 16 16	7 6 4	8 8 8	450 269 200
Time window 0	206 1,210	1,298	27,000	175 98 64	14 14 14	6 5 4	7 7 7	202 124 89
Time window 1	150 952	939	20,000	123 73 54	15 15 14	5 5 4	6 6 6	149 99 78
Time window 2	175 897	1,166	24,000	140 95 65	15 15 15	6 5 4	6 6 7	167 121 91

the *Dimension Reduction* and *Clustering* stages, but it is rather difficult to achieve increasing improvements because these runtimes are already very low. The *Outlier Detection* stage has been omitted as its runtimes are the lowest ones (close to 1 second) due to its relatively simple complexity, and the values measured were mostly the same for all experiments. Note that the local process-based parallelism serves as a simple first approach for the processing stages required after *Feature Extraction*. However, these stages should not pose any scalability issue as their complexity is significantly lower, as analysed in Section 3.3 (see Table 3).

5. Conclusions

Time series are present in many fields of study, as they represent any kind of numerical measurement along time, which in essence boils down to characterizing a behaviour. The study of time series has already been extensively discussed in the literature regarding their mathematical properties, but more recently also computer scientists have found in time series a potential source of valuable data to be exploited using new available tools and technologies such as Machine Learning and Big Data. A common use case is the classification of existing and previously labelled time series to create groups that allow to quickly identify new time series. However, in those scenarios where there is no available information, it is also possible to cluster the time series to create groups, albeit such groups can or cannot be meaningful, as no labelled data can be used for validation. In addition, in some use cases several time series can be combined (i.e., they are multivariate) to model the behaviour of more complex entities, although usually this is at the expense of an increased overall difficulty.

In this paper we have presented a practical example of such multivariate scenarios that focuses on HPC jobs from several users as the entities to be studied, using their multiple computing nodes, as well as their multiple resource time series per node as data. With these series we opted to apply clustering of the jobs by using only extracted information from them, without depending on any type of labelling or

previous classification. Such clustering has been implemented with a multi-process pipeline that uses feature extraction of the time series, along with an outlier detection, a dimension reduction and a clustering, all in a highly configurable manner to ultimately perform a multi-predictor search of the most promising result in a last heuristic-guided evaluation stage. In addition, the outlier jobs detected have been studied as the basis for an anomaly detection scenario, as well as for further clustering if too many outlier and potentially anomalous jobs are produced. The computational complexity and the scalability of our solution has also been analysed, considering that its technical implementation relies on the fusion of ML and Big Data technologies such as Apache Spark, and that its design is based on a pipeline architecture and transformer operations.

To prove our proposed pipeline, several experiments have been conducted covering different use cases: from a single user and a few job types (medium variability) to those with several users and several expected job types (high variability). Both evaluation metrics and visualization techniques have been provided and used for each experiment. They aim at assessing numerically whether a predictor provides a promising clustering option, as well as at guiding users (e.g., system administrators) in quickly performing a visual evaluation. With these tools in hand we showed how it is possible to process datasets containing hundreds of jobs, each composed of many node and resource time series, in order to extract groups that relate to different job patterns or to different users. Furthermore, we have shown that outlier detection can be optionally performed on a dataset to isolate anomalous jobs or groups of jobs that may have suffered some kind of error.

Finally, it has to be considered that any unsupervised scenario presents some intrinsic limits regarding the potential to have a good separation of the jobs and to evaluate the fitness of such separation, which arise from the lack of labelled data. Nevertheless, these scenarios do occur in practice, and it is necessary to continue working on systems such as our pipeline that can efficiently handle numerous multivariate time series and extract valuable information from them in the absence of external information.

The source code of all the programs and scripts, as well as all the data used in this work (datasets with the metrics and features, plots, generated files with the evaluation metrics…), are publicly available in a repository hosted in Mendeley Data [57].

CRediT authorship contribution statement

Jonatan Enes: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Roberto R. Expósito:** Validation, Writing – review & editing, Visualization. **José Fuentes:** Conceptualization, Methodology. **Javier López Cacheiro:** Conceptualization, Methodology, Software, Validation, Resources, Supervision. **Juan Touriño:** Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The link to the Mendeley repository containing the source code and the datasets is available at the end of the article.

Datasets and source code for a pipeline architecture for feature-based unsupervised clustering using multivariate time series from HPC jobs (Original data) (Mendeley Data)

Acknowledgements

This research was funded by the Ministry of Science and Innovation of Spain (PID2019-104184RB-I00/AEI/10.13039/501100011033), and by Xunta de Galicia, Spain and FEDER funds of the European Union (Centro de Investigación de Galicia accreditation 2019–2022, ref. ED431G 2019/01; Consolidation Program of Competitive Reference Groups, ref. ED431C 2021/30). Funding for open access charge: Universidade da Coruña/CISUG.

References

[1] S. Aghabozorgi, A.S. Shirkhorshidi, T.Y. Wah, Time-series clustering – a decade review, Inf. Syst. 53 (2015) 16–38.
[2] V. Bolón-Canedo, A. Alonso-Betanzos, Ensembles for feature selection: A review and future trends, Inf. Fusion 52 (2019) 1–12.
[3] CESGA supercomputing centre, 2022, https://www.cesga.es, Last visited: December 2022.
[4] B. Zhao, H. Lu, S. Chen, J. Liu, D. Wu, Convolutional neural networks for time series classification, J. Syst. Eng. Electron. 28 (1) (2017) 162–169.
[5] C.-L. Liu, W.-H. Hsaio, Y.-C. Tu, Time series classification with multivariate convolutional neural network, IEEE Trans. Ind. Electron. 66 (6) (2019) 4788–4797.
[6] Y. Zheng, Q. Liu, E. Chen, Y. Ge, J.L. Zhao, Exploiting multi-channels deep convolutional neural networks for multivariate time series classification, Front. Comput. Sci. 10 (1) (2016) 96–112.
[7] J. Lines, A. Bagnall, Time series classification with ensembles of elastic distance measures, Data Min. Knowl. Discov. 29 (3) (2015) 565–592.
[8] T. Górecki, M. Łuczak, Multivariate time series classification with parametric derivative dynamic time warping, Expert Syst. Appl. 42 (5) (2015) 2305–2312.
[9] A. Bagnall, J. Lines, J. Hills, A. Bostrom, Time-series classification with COTE: The collective of transformation-based ensembles, IEEE Trans. Knowl. Data Eng. 27 (9) (2015) 2522–2535.
[10] F.J. Baldán, J.M. Benítez, Distributed fastshapelet transform: A big data time series classification algorithm, Inform. Sci. 496 (2019) 451–463.
[11] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, Apache spark: A unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65.
[12] M.F. Ghalwash, Z. Obradovic, Early classification of multivariate temporal observations by extraction of interpretable shapelets, BMC Bioinformatics 13 (195) (2012) 1–12.
[13] B.D. Fulcher, N.S. Jones, Highly comparative feature-based time-series classification, IEEE Trans. Knowl. Data Eng. 26 (12) (2014) 3026–3037.
[14] A. Zagorecki, A versatile approach to classification of multivariate time series data, in: Proceedings of 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, Łódź, Poland, 2015, pp. 407–410.
[15] P. Schäfer, The BOSS is concerned with time series classification in the presence of noise, Data Min. Knowl. Discov. 29 (6) (2015) 1505–1530.
[16] N. Tavakoli, S. Siami-Namini, M.A. Khanghah, F.M. Soltani, A.S. Namin, An autoencoder-based deep learning approach for clustering time series data, SN Appl. Sci. 2 (5) (2020) 1–25.
[17] T.W. Liao, Clustering of time series data – a survey, Pattern Recognit. 38 (11) (2005) 1857–1874.
[18] S.-E. Benkabou, K. Benabdeslem, B. Canitia, Unsupervised outlier detection for time series by entropy and dynamic time warping, Knowl. Inf. Syst. 54 (2) (2018) 463–486.
[19] H. He, Y. Tan, Unsupervised classification of multivariate time series using VPCA and fuzzy clustering with spatial weighted matrix distance, IEEE Trans. Cybern. 50 (3) (2018) 1096–1105.
[20] J. Zakaria, A. Mueen, E. Keogh, Clustering time series using unsupervised-shapelets, in: Proceedings of the 12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, 2012, pp. 785–794.
[21] G. Anand, R. Nayak, Unsupervised visual time-series representation learning and clustering, in: Proceedings of the 27th International Conference on Neural Information Processing, ICONIP 2020, Bangkok, Thailand, Online, 2020, pp. 832–840.
[22] C.T. Zan, H. Yamana, An improved symbolic aggregate approximation distance measure based on its statistical features, in: Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services, IiWAS '16, Singapore, 2016, pp. 72–80.
[23] Y. Yu, Y. Zhu, D. Wan, H. Liu, Q. Zhao, A novel symbolic aggregate approximation for time series, in: Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication, IMCOM 2019, Phuket, Thailand, 2019, pp. 805–822.
[24] L. Wang, F. Lu, M. Cui, Y. Bao, Survey of methods for time series symbolic aggregate approximation, in: Proceedings of the 5th International Conference of Pioneering Computer Scientists, Engineers and Educators, ICPCSEE 2019, Guilin, China, 2019, pp. 645–657.

[25] J. Hartung, G. Gühring, V. Licht, A. Warta, Comparing multidimensional sensor data from vehicle fleets with methods of sequential data mining, SN Appl. Sci. 2 (4) (2020) 1–13.
[26] M.S. Halawa, R.P. Díaz Redondo, A. Fernández Vilas, Unsupervised KPIs-based clustering of jobs in HPC data centers, Sensors 20 (15) (2020) 4111:1–4111:21.
[27] D. Tiano, A. Bonifati, R. Ng, FeatTS: Feature-based time series clustering, in: Proceedings of the 2021 International Conference on Management of Data, SIGMOD/PODS'21, Xi'an, Shaanxi, China, Online, 2021, pp. 2784–2788.
[28] B.D. Fulcher, N.S. Jones, Hctsa: A computational framework for automated time-series phenotyping using massive feature extraction, Cell Syst. 5 (5) (2017) 527–531.
[29] M. Christ, N. Braun, J. Neuffer, A. Kempa-Liehr, Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package), Neurocomputing 307 (2018) 72–77.
[30] C.H. Lubba, S. Sethi, P. Knaute, S. Schultz, B. Fulcher, N. Jones, catch22: CAnonical Time-series CHaracteristics, Data Min. Knowl. Discov. 33 (2019) 1821–1852.
[31] D. Shaykhislamov, V. Voevodin, An approach for dynamic detection of inefficient supercomputer applications, Procedia Comput. Sci. 136 (2018) 35–43.
[32] L. Erhan, M. Ndubuaku, M. Di Mauro, W. Song, M. Chen, G. Fortino, O. Bagdasar, A. Liotta, Smart anomaly detection in sensor systems: A multi-perspective review, Inf. Fusion 67 (2021) 64–79.
[33] N. Laptev, S. Amizadeh, I. Flint, Generic and scalable framework for automated time-series anomaly detection, in: Proceedings of 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'15, Sydney, Australia, 2015, pp. 1939–1947.
[34] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, Anomaly detection using autoencoders in high performance computing systems, in: Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI-19, Honolulu, HI, USA, 2019, pp. 9428–9433.
[35] M. Erz, J.F. Kielman, B.S. Uzun, G.S. Gühring, Anomaly detection in multidimensional time series – a graph-based approach, J. Phys. Complex. 2 (4) (2021) 045018.
[36] M. Çelik, F. Dadaşer-Çelik, A. Ş. Dokuz, Anomaly detection in temperature data using DBSCAN algorithm, in: Proceedings of the 2011 International Symposium on INnovations in Intelligent SysTems and Applications, INISTA 2011, Istanbul, Turkey, 2011, pp. 91–95.
[37] G. Ozer, A. Netti, D. Tafani, M. Schulz, Characterizing HPC performance variation with monitoring and unsupervised learning, in: Proceedings of the 35th International Conference on High Performance Computing, ISC 2020, Frankfurt, Germany, 2020, pp. 280–292.
[38] B. Wang, Z. Mao, Outlier detection based on a dynamic ensemble model: Applied to process monitoring, Inf. Fusion 51 (2019) 244–258.
[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[40] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, et al., API design for machine learning software: Experiences from the scikit-learn project, in: European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases, Languages for Data Mining and Machine Learning Workshop, ECML/PKDD 2013, Prague, Czech Republic, 2013, pp. 108–122.
[41] A. Svyatkovskiy, K. Imai, M. Kroeger, Y. Shiraito, Large-scale text processing pipeline with apache spark, in: Proceedings of the 2016 IEEE International Conference on Big Data, IEEE BigData 2016, Washington D.C., USA, 2016, pp. 3928–3935.
[42] M. Liu, Z. Xue, X. He, A unified host-based intrusion detection framework using spark in cloud, in: Proceedings of the IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications, TrustComp 2020, Guangzhou, China, 2020, pp. 97–103.
[43] W. McKinney, Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, O'Reilly Media, 2012.
[44] A.B. Yoo, M.A. Jette, M. Grondona, Grondona, slurm: Simple linux utility for resource management, in: Proceedings of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP 2003, Seattle, WA, USA, 2003, pp. 44–60.
[45] A. Komarek, J. Pavlik, L. Mercl, V. Sobeslav, Metric based cloud infrastructure monitoring, in: Proceedings of the 12th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2017, Barcelona, Spain, 2017, pp. 391–400.
[46] T.W. Wlodarczyk, Overview of time series storage and processing in a cloud environment, in: Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science, CloudCom'12, Taipei, Taiwan, 2012, pp. 625–628.
[47] The apache software foundation, HBase: A distributed database for large datasets, 2022, https://hbase.apache.org, Last visited: December 2022.
[48] Tsfresh list of extraction functions, 2022, https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html, Last visited: December 2022.
[49] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al., MLlib: Machine learning in Apache Spark, J. Mach. Learn. Res. 17 (2016) 34:1–34:7.
[50] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu, Understanding of internal clustering validation measures, in: Proceedings of the 10th IEEE International Conference on Data Mining, ICDM 2010, Sydney, Australia, 2010, pp. 911–916.
[51] P.J. Rousseeuw, Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, J. Comput. Appl. Math. 20 (1987) 53–65.
[52] D.L. Davies, D.W. Bouldin, A cluster separation measure, IEEE Trans. Pattern Anal. Mach. Intell. (2) (1979) 224–227.
[53] T. Caliński, J. Harabasz, A dendrite method for cluster analysis, Comm. Statist. Theory Methods 3 (1) (1974) 1–27.
[54] B.C. Kwon, B. Eysenbach, J. Verma, K. Ng, C. De Filippi, W.F. Stewart, A. Perer, Clustervision: Visual supervision of unsupervised clustering, IEEE Trans. Vis. Comput. Graphics 24 (1) (2018) 142–151.
[55] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, E. Baldeschwieler, Apache hadoop YARN: Yet another resource negotiator, in: Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC'13, Santa Clara, CA, USA, 2013, pp. 5:1–5:16.
[56] P. Singh, Data processing, in: Learn PySpark, A Press, 2019, pp. 17–48.
[57] Datasets and source code for a pipeline architecture for feature-based unsupervised clustering using multivariate time series from HPC jobs, mendeley data, 2022, http://dx.doi.org/10.17632/hgkv9cpnmn.2.