Journal of Parallel and Distributed Computing 189 (2024) 104881

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

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

A novel framework for generic Spark workload characterization and similar pattern recognition using machine learning

Mariano Garralda-Barrio *, Carlos Eiras-Franco, Verónica Bolón-Canedo

CITIC, Universidade da Coruña, A Coruña, Spain

ARTICLE INFO

Dataset link: https://github.com/mgarralda/hadoop-spark-cluster/tree/main/spark-event-logs

Keywords:
Big data
Workload characterization
Apache spark
Pattern recognition
Machine learning

ABSTRACT

Comprehensive workload characterization plays a pivotal role in comprehending Spark applications, as it enables the analysis of diverse aspects and behaviors. This understanding is indispensable for devising downstream tuning objectives, such as performance improvement. To address this pivotal issue, our work introduces a novel and scalable framework for generic Spark workload characterization, complemented by consistent geometric measurements. The presented approach aims to build robust workload descriptors by profiling only quantitative metrics at the application task-level, in a non-intrusive manner. We expand our framework for downstream workload pattern recognition by incorporating unsupervised machine learning techniques: clustering algorithms and feature selection. These techniques significantly improve the process of grouping similar workloads without relying on predefined labels. We effectively recognize 24 representative Spark workloads from diverse domains, including SQL, machine learning, web search, graph, and micro-benchmarks, available in HiBench. Our framework achieves a high accuracy F-Measure score of up to 90.9% and a Normalized Mutual Information of up to 94.5% in similar workload pattern recognition. These scores significantly outperform the results obtained in a comparative analysis with an established workload characterization approach in the literature.

1. Introduction

With the volume of data increasing exponentially, the era of big data has emerged as one of the most significant trends in high-performance computing. To extract valuable insights, big data workloads demand specialized environments on large-scale computing infrastructures. To address this requirement, Apache Spark™ [1] is a widely-used framework that provides a unified multi-language engine for computing heterogeneous workloads, such as data engineering, data analytics, and machine learning applications. Comprehensive workload characterization is essential for understanding Spark applications. This, in turn, facilitates the development of downstream objectives, such as performance prediction models [11], workload prediction [25], and auto-tuning of big data applications [33], allowing for proactive optimization of resource allocation. However, the distributed nature of Spark infrastructure (it), large datasets (ds), diverse application characteristics (app), and numerous configuration settings (cs) present significant challenges in characterizing Spark workloads.

As a definition, workload characterization [24] consists of a description of the workload by means of several quantitative metrics, such as at micro-architecture-level, system-level, and application-level. Nevertheless, most ongoing efforts in workload characterization tend to focus on system-level properties, which include machine-specific details, rather than pure workload characterization. This would involve understanding the behavior and patterns of a workload without making any assumptions about the underlying system or hardware. Likewise, a generic characterization adequately describes any type of workload without focusing on any specific application or domain. A workload characterization derives a feature descriptor that is capable of reproducing its behavior. However, it is difficult to define a suitable workload descriptor in Spark due to the complexity of data processing, computation and communication patterns. Furthermore, tens to hundreds of Spark properties are involved, which interact with each other in several complex ways.

In general, we can identify two types of metrics to describe distributed applications: statics and dynamics. Static metrics of the workloads describe the inherent characteristics of the application such as data properties and application settings. On the other side, dynamic metrics describe the behavior of an application over time when it is executed on a given system. Thus, resource metrics can be presented as a

* Corresponding author.
E-mail addresses: mariano.garralda@udc.es (M. Garralda-Barrio), carlos.eiras@udc.es (C. Eiras-Franco), veronica.bolon@udc.es (V. Bolón-Canedo).

https://doi.org/10.1016/j.jpdc.2024.104881
Received 6 June 2023; Received in revised form 22 January 2024; Accepted 18 March 2024

Available online 26 March 2024

time-series, which is a sequence of values typically measured at successive points in time spaced. The approach commonly adopted for workload characterization [40] is based on the analysis of metrics collected on the system while the application is running. Appropriate instrumentation for data collection should be developed to ensure the quality for workload profiling measurements. The degree of intrusiveness and the overhead introduced by the instrumentation system have to be as low as possible in order not to perturb the behavior of the system and of its workload.

Empirical Spark workload models can be built using supervised machine learning techniques, as demonstrated by previous research [16]. However, creating accurate models for workload similarity analysis at a low cost can be a challenging and time-consuming process. This is mainly attributed to the challenges of accurately labeling training data from specific environments, such as large hardware infrastructure and multi-layer software stacks (e.g., databases, distributed file systems), which arise from the intricate interdependencies among these components. Moreover, supervised models are limited to recognizing only labeled workloads present in the training set, which can result in incorrect identification of previously unseen workloads. Therefore, further research is needed to improve the scalability and accuracy of supervised models for workload similarity analysis, especially in dynamic environments where new workloads are frequently executed.

Unsupervised learning techniques are a viable alternative to supervised machine learning for recognizing previously unseen workloads, as they do not require semantic labeling. Clustering techniques [23,39,22] have been widely used in Spark applications to identify similar workloads by grouping them based on similar behavior. This simplifies the building of models for classification, performance, and optimization, like those presented in [25], [36], and [27], respectively. To validate clustering models through specialized metrics, experimental labeled datasets are usually used, and these involve assigning ground truth classes to the data for evaluation purposes. In that sense, benchmarking is a useful approach for workload evaluation models. One widely used benchmark for big data processing systems is HiBench [21], which includes a set of workloads that cover various big data processing functions. These workloads are representative of real-world domains, and can be used to evaluate Spark applications under different conditions, such input data sizes and application settings.

1.1. Motivation

In big data analytics, applications often prioritize exploration and experimentation, resulting in less repetitive workloads compared to traditional small data systems. As a result, in Spark production infrastructures, multiple types of workloads are run on a daily basis, each with different application settings (optimal or suboptimal), and/or input/output data sizes. However, lack of sufficient prior information often leads to unknown underlying execution behaviors of workloads, making it challenging to recognize patterns among various Spark applications. This characterization is essential for downstream activities like identifying workload similarities and ultimately contributes to subsequent Spark-based optimization efforts.

1.2. Our contribution

This study introduces a novel framework for effective characterizes Spark workloads, serving as the foundation for downstream pattern recognition. Our approach represents each set of Spark task metrics as time-series, enabling in-depth analysis of their temporal behavior. To characterize the workloads, we apply cumulative sum transformations and higher-order polynomial regressions. This feature engineering process enables the construction of interpretable workload descriptors (latent vectors) and enhances domain understanding compared to other black-box approaches.

To identify similar patterns, we employ machine learning techniques, including both partitional and hierarchical clustering, as well as univariate and multivariate filters for unsupervised feature selection. This significantly enhances our ability to recognize Spark workloads within the HiBench as experimental validation results. These workloads encompass representative domains such as SQL, machine learning, web search, graph processing, and micro-benchmarks. We run each workload with different application settings on several input data sizes to analyze the variability of the workload descriptors. In summary, the main contributions of this paper are as follows:

- We propose a novel framework for characterizing Spark applications by building consistent workload descriptors in a generic, individual and scalable manner.
- We establish downstream similar pattern recognition of workloads using unsupervised machine learning techniques.
- We analyzed 24 representative Spark workloads from HiBench across several domains and evaluated them using a range of clustering metrics, including geometric, internal, and external measurements.

The rest of the paper is organized as follows. Section 2 describes the background. Section 3 introduces the related work. Section 4 depicts the methodology of the framework. Section 5 explains the experimental design. Section 6 describes the results for workload characterization and similar pattern recognition. Section 7 assess the practical applicability of the framework. Section 8 concludes the paper.

2. Background

In short, Spark applications run on the Driver, which is a specific node within the cluster not participating in distributed computation. At software level, Spark has two main operations: *transformations* executed in lazy mode and *actions* immediately executed, such as *count()*, *read()* or *write()*. Those operations are applied on resilient distributed dataset (RDD) across computation nodes (workers) of the cluster. Spark has built-in optimizers to improve execution plans based on the type of workload (e.g., SparkSQL compile SQL plans).

Internally in Spark (Fig. 1), a workload consists of many jobs, each of which is represented as a directed acyclic graph (DAG). A job is a sequence of stages triggered by an *action*. A stage is a sequence of tasks that not require a shuffle (data re-partitioned across the computer nodes) in-between, that is, they can all be run in parallel without a shuffle. Depending on the job, stages could be running in parallel. Every node in a DAG represents a task, which is a single operation applied to a single partition (logical chunk of dataset). Each task is executed by single thread (core) in an Executor (JVM running in Spark worker node). This means that the Spark framework considers tasks as the smallest unit of parallelism. Thus, by focusing on application task-level metrics for Spark workload characterization, we can gain a suitable understanding of their behavior.

Among the pivotal Spark parameters that influence parallelism, key contenders include the count of *executors*, the allocation of *executorCores*, the memory allocation for each *executor*, and the strategic *shufflePartitions*. The latter parameter influences workloads with SparkSQL operations by determining the partition count used in data shuffling. This is a fundamental aspect of transformations like grouping, joining, and aggregation.

Running a given Spark workload n times, the generated DAGs are deterministic. This means that the behavior of stages (actions and transformations) and their logical parallelism remains the same. However, the actual number of tasks running in parallel depends on the number of cores available in the system as well as the Spark application settings. Fig. 2 shows two executions of the same Spark workload (wordcount) with different core settings (1 vs 8 cores). Examining the timeline of the

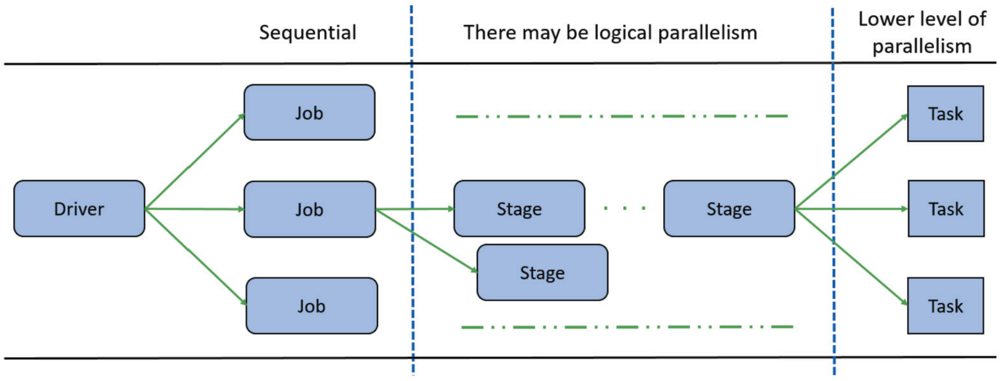


Fig. 1. Architecture plan for executing Spark workload application.

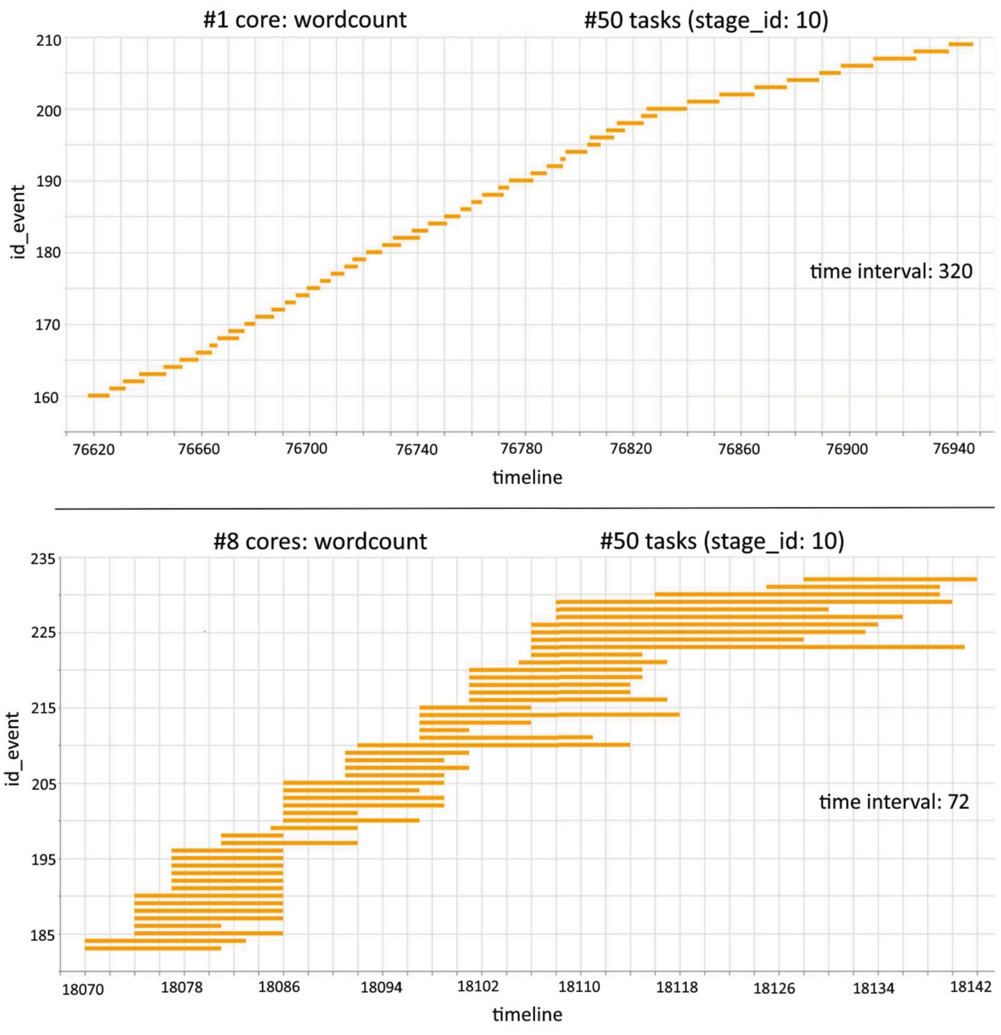


Fig. 2. Window timeline of stage tasks for two Spark wordCount executions, using 1 and 8 cores, respectively.

respective stages, we can observe that the number of tasks executed in parallel varies, which strongly affects performance.

Spark provides a REST API [2] to access task metrics collected by executors during task execution with fine-grained granularity. These metrics can be accessed at runtime or just finished, and an ordered sequence of internal event logs is also available as a downloadable zipped *SparkEventLog* file. Task-level metrics, such as shuffling, executors, deserialization, and serialization time, offer valuable insights into the behavior of specific tasks in a Spark workload. For instance, metrics like *'shuffle write time'*, *'shuffle bytes written'*, and *'shuffle records written'* help identify tasks with high shuffling overheads. Similarly, *'executor CPU'* time and *'executor deserialize time'* help identify computationally intensive tasks or those experiencing significant network serialization. By analyzing these task-level metrics as features, we can gain valuable insights into the patterns of Spark workloads.

If $\alpha, \beta, \gamma, \dots \omega$ is a list of n features that completely describe a workload (W), we can represent it as a workload descriptor (W_d) by applying a function (f), as follows:

$$W = f(it, ds, app, cs) \xrightarrow[\text{characterization}]{\text{workload}} [\alpha, \beta, \gamma, \dots \omega] = W_d, \quad (1)$$

therefore, a matrix $(m \cdot n)$ represents a set of workload descriptors $(\mathbb{W}_{m,n})$ in Eq. (2). This matrix serves as a foundation for feature-based approach solutions. Decoupling feature extraction from the downstream objectives offers flexibility and reusability.

$$\mathbb{W}_{m,n} = \begin{bmatrix} \alpha_1 & \beta_1 & \gamma_1 & \cdots & \omega_1 \\ \alpha_2 & \beta_2 & \gamma_2 & \cdots & \omega_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_m & \beta_m & \gamma_m & \cdots & \omega_m \end{bmatrix} = \begin{bmatrix} W_{d_1} \\ W_{d_2} \\ \vdots \\ W_{d_m} \end{bmatrix} \tag{2}$$

Based on the workload characterization function defined in Eq. (1), our framework treats the infrastructure (it) as a fixed parameter. Consequently, we concentrate on the remaining arguments: the dataset (ds), the application characteristics (app), and the configuration settings (cs).

3. Related work

Characterizing Spark workloads is a fundamental step for subsequent domain activities, including optimization, performance modeling, and pattern recognition—of which the latter is a key part of our research. However, prior studies have not adequately emphasized the core aspect of generic workload characterization, leading to an insufficiency in the existing literature. In this section, we provide an overview of related studies, establishing a conceptual link to our work.

One of the first papers that addressed Spark workloads [13], proposed a method for characterizing TPC-H queries based on system-level metrics, such as I/O, memory and CPU usage, to optimize query execution on Apache Spark. In their work [14], the authors also rely on system-level metrics for JVM characterization, specifically designed for machine learning workloads in Spark. Their approach involves the utilization of statistical methods to thoroughly analyze and optimize the workload. The work presented in [8] focuses on the micro-architectural characterization of Apache Spark, analyzing the computation and communication patterns of the workload. The authors in this study use metrics such as cache hit rate, memory bandwidth, and CPU utilization. Nevertheless, characterizing Spark workloads based solely on system-level or micro-architectural metrics may not provide a comprehensive understanding of the application's behavior, especially for complex workloads that involve iterative processing or large-scale machine learning models.

Performance analysis offers an additional approach to characterizing Spark workloads. In recent years, there has been an increasing interest in using machine learning techniques to develop performance models to predict the execution time of Spark workloads. Such models can assist users in optimizing resource allocation and enhancing application performance. For example, in [40], they utilize micro-architectural and application job-level metrics for workload characterization, and machine learning techniques to identify the most important metrics that affect the performance of big data workloads. In [30], the authors use a combination of system-level and application stage-level metrics to evaluate the performance of Spark workloads. One limitation of this proposal is its lack of generalizability to all Spark workloads, as the set of extracted features may only be applicable to SparkSQL applications. The research work presented in [27] proposed, in terms of characterization, a combination of system-level metrics (e.g., CPU utilization, memory usage, disk I/O) and application-specific metrics (e.g., input data size, number of RDD partitions, number of stages). The weakness of this study is that it requires access to the application code and its specific metrics, which may not be available in some scenarios. Additionally, the work assumes a linear relationship between the metrics and the application performance, which may not always hold in practice.

To further explore the theme of workload characterization through performance models, a study by [36] leverages a range of features including input data size, number of tasks, and memory usage to develop a performance prediction model. Specifically, the authors employ a random forest regression algorithm to train the model, enabling accurate predictions of the execution time of Spark workloads. However, they only evaluate their approach on a limited set of benchmark applications, and it is unclear how well their technique generalizes to other types of Spark applications. Another performance model is presented in [12] by leveraging a subset of important features from a large set of profiling metrics. The authors suggest a feature selection algorithm based on a combination of statistical techniques and domain knowledge to identify the most relevant metrics for prediction.

Pattern recognition is an important aspect of Spark workload characterization that can help in understanding the behavior of complex systems. For instance, in the study conducted by [17], a machine learning-based Spark and Hadoop workload classification was proposed based on container performance patterns. They demonstrated the effectiveness of their proposal by accurately classifying Spark and Hadoop workloads based on their performance characteristics. In [25], the authors proposed a semantic-aware method to workload characterization and consolidation in cloud data centers. They argue that traditional workload characterization techniques rely on manually defined performance metrics that may not accurately reflect the behavior of complex workloads. Their study uses semantic information about the tasks performed by the workloads to automatically identify their characteristics and consolidate them onto shared resources. The path involves using a combination of machine learning and semantic reasoning techniques to capture the workload semantics and make informed consolidation decisions. Another study [25] conducted by the same authors presents Phase Annotated Learning (PAL) technique suggested for workload profiling, detection and resource usage prediction for Spark workloads. PAL leverages phase annotation to capture the performance behavior of Spark jobs and uses clustering algorithms to identify similar workload patterns. To achieve this, they introduce a novel machine learning framework that uses annotated execution traces to automatically recognize and classify Spark workload phases. However, the workload characterization results pertain specifically to a downstream objective, thus limiting the overall generalizability of the approach.

Overall, profiling dynamic metrics can provide a comprehensive understanding of Spark workload execution. Machine learning techniques offer more flexibility and generalization capabilities compared to traditional metrics-based mechanisms. Although supervised machine learning models can capture complex relationships, they require a significant amount of labeled data to train, which may not be readily available in real Spark environments. Thereby, our framework eliminates the need for additionally semantic information in the workload characterization process and downstream similar pattern recognition. Furthermore, our approach is scalable to new Spark environments and previously unencountered applications since it generates workload descriptors with both a generic and individual nature.

4. Methodology framework

In this section, we present our framework for generic Spark workload characterization and the goal of downstream pattern recognition. A visual overview of the proposed framework is presented in Fig. 3. Our emphasis is on feature engineering to construct interpretable descriptors, thereby characterizing workloads in latent space. Subsequently, we delve into cluster algorithms employed for the pattern recognition. Lastly, we employ unsupervised feature selection algorithms to rank the most relevant features, thus enhancing this goal of the framework.

4.1. Workload characterization phase

The first phase of our framework focuses on the collection and profiling of task execution metrics. Subsequently, we employ a range of transformations techniques to derive meaningful workload descriptors.

4.1.1. Data collecting and profiling

Firstly, we commence by examining an arbitrary set (N) of specific events within the *SparkEventLog* file, such as jobs, stages, and tasks.

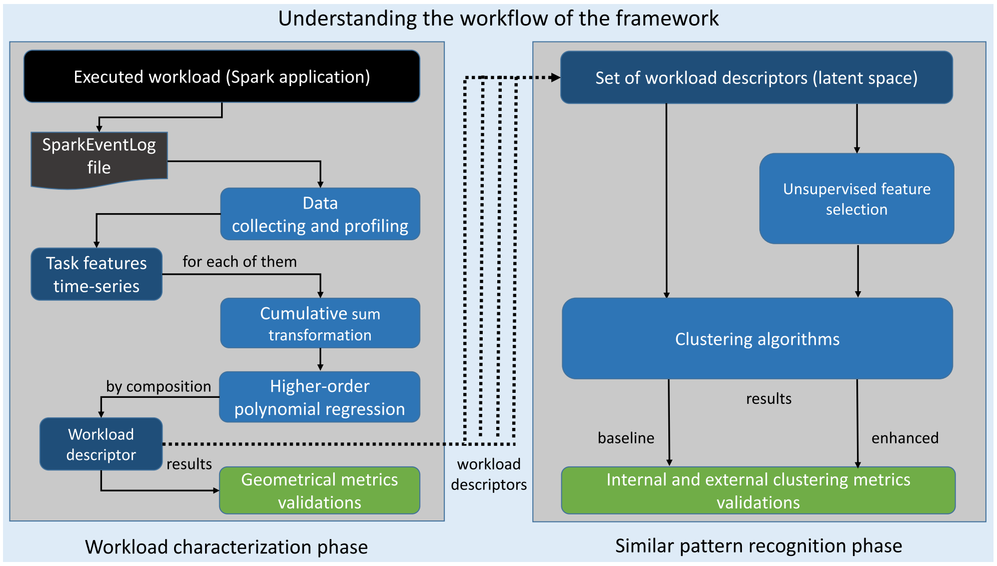

Understanding the workflow of the framework
Executed workload (Spark application)
SparkEventLog file
Data collecting and profiling
Task features time-series
for each of them
Cumulative sum transformation
by composition
Higher-order polynomial regression
Workload descriptor
results
Geometrical metrics validations
workload descriptors
Set of workload descriptors (latent space)
Unsupervised feature selection
Clustering algorithms
results
baseline
enhanced
Internal and external clustering metrics validations
Workload characterization phase
Similar pattern recognition phase

Fig. 3. The proposed workflow of our framework for generic Spark workload characterization and downstream similar pattern recognition.

Table 1
Executor task metrics, as delineated within the Spark monitoring and instrumentation documentation [2].

Metrics at the task-level	
1 Executor Run Time	13 Output: Bytes Written
2 Executor CPU Time	14 Output: Records Written
3 Executor Deserialize Time	15 Shuffle Read: Records Read
4 Executor Deserialize CPU Time	16 Shuffle Read: Remote Blocks Fetched
5 Result Size	17 Shuffle Read: Local blocks Fetched
6 JVM GC Time	18 Shuffle Read: Remote Bytes Read
7 Result Serialization Time	19 Shuffle Read: Local Bytes Read
8 Memory Bytes Spilled	20 Shuffle Read: Remote Bytes Read Disk
9 Disk Bytes Spilled	21 Shuffle Read: Fetch Wait Time
10 Peak Execution Memory	22 Shuffle Write: Bytes Written
11 Input: Bytes Read	23 Shuffle Write: Records Written
12 Input: Records Read	24 Shuffle Read: Write Time

Upon observation, it becomes evident that each individual event is allocated a synchronized sequential id number, in accordance with the formulation depicted in Eq. (3) Apache Spark achieves synchronization of time exposed in log files by utilizing a common reference time, leveraging system-level time synchronization mechanisms, and consistently timestamping log events. This ensures that log events from different workers and executors can be accurately ordered and analyzed in the context of a Spark application.

$$(event_{id} \xrightarrow{\text{starts at}} time_{x}) \Rightarrow (event_{id+1} \xrightarrow{\text{starts at}} time_{x'}),\quad \forall_{id} \in \{0 \dots \mathrm{N}-1\},\ x \le x' \tag{3}$$

To create a Spark workload descriptor, we exclusively analyze and profile task execution metrics collected from the specified log file once the application has finished. The complete roster of task metrics available in the Spark version employed in this study, is displayed in Table 1. Our selection of these metrics is based on domain-specific comprehension. Each task metric corresponds to a crucial low-level execution aspect of Spark applications, which clarifies the systematic logging of these metrics by Spark. In prior research [33], the authors utilized these types of metrics to characterize Spark workloads, computing several statistical summaries for each one.

A significant aspect of our study is our ability to represent the values of each task metric as a time-series. This allows us to obtain valuable insights into their behavior throughout the execution life cycle, extending beyond basic statistical characterization. Consequently, each task metric servers as an independent attribute (Task-feature) contributing to the composition of the workload descriptor. To enhance understanding, we provide an illustrative example by partially examining a specific *SparkEventLog* from an executed Spark workload. The log file (Fig. 4) accommodates task events, notably the *SparkListenerTaskStart* and *SparkListenerTaskEnd* events, which are recorded sequentially. The number of entries in these events (Task ID) essentially corresponds to the total count of tasks executed by the application across the employed executors. When we focus on a *SparkListenerTaskEnd* event, we observe that we can extract a set of task metrics, as delineated in Table 1. Hence, each task metric can be characterized as a time-series of values that aligns with Eq. (3), contributing to the composition of the Task-features.

Up to this point, a resulting descriptor consists of a matrix ($\mathbb{TF}$) comprised of Task-features (Tf) time-series, each of which is represented by task metric values (tf). Its dimensions are determined by the number of task metrics (m) and the total count of tasks (n) generated by a specific Spark workload, as illustrated in Eq. (4).

$$\mathbb{TF}_{m,n} = \begin{bmatrix} tf_{1,1} & tf_{1,2} & tf_{1,3} & \cdots & tf_{1,n} \\ tf_{2,1} & tf_{2,2} & tf_{2,3} & \cdots & tf_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ tf_{m,1} & tf_{m,2} & tf_{m,2} & \cdots & tf_{m,n} \end{bmatrix} = \begin{bmatrix} Tf_1 \\ Tf_2 \\ \vdots \\ Tf_m \end{bmatrix} \tag{4}$$

Importantly, we employ a non-standard min-max scaling approach on individual Tf time-series data. This approach prioritizes the modeling of behavior patterns over absolute statistical summaries of task metric values. The process is as follows:

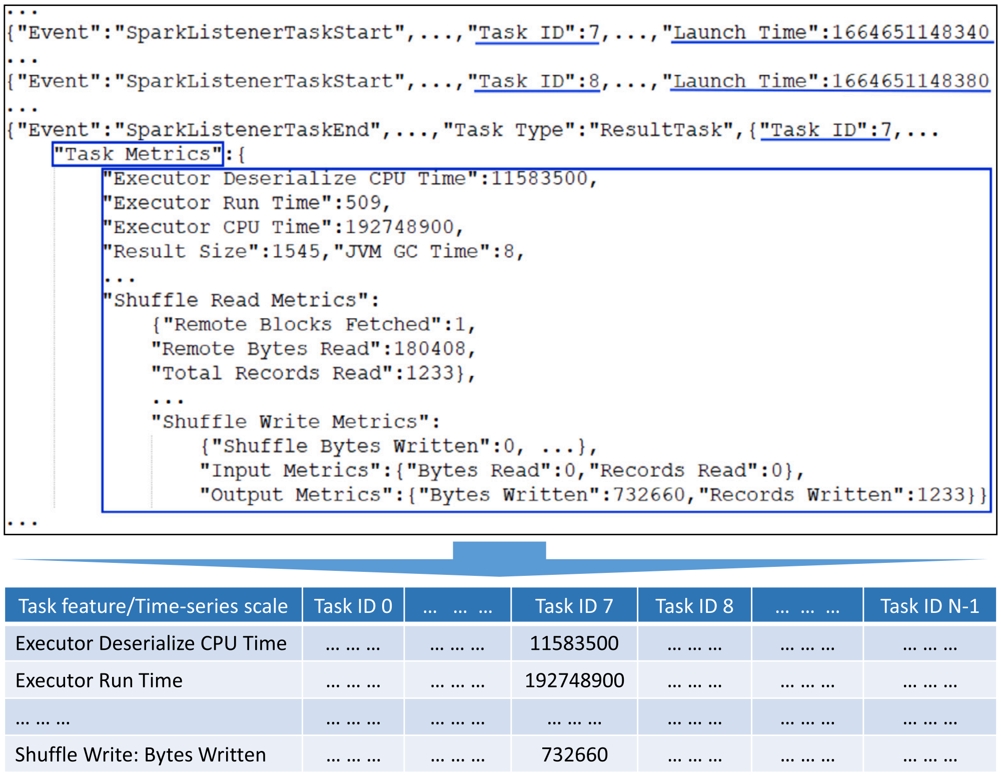
$$tf'_{m,n} = \frac{tf_{m,n} - min(Tf_{m,:})}{max(Tf_{m,:}) - min(Tf_{m,:})}, \tag{5}$$

where $min(Tf_{m,:})$ and $max(Tf_{m,:})$ are the minimum and maximum values of the m^{th} time-series of $\mathbb{TF}$, respectively.

Fig. 5 illustrates a task time-series metric for varying input data sizes in a specific Spark workload. It's important to highlight that the distributions in the figures exhibit patterns in the timing and manner of data writing. This enables us to establish mechanisms for identifying workloads that exhibit similar behavior, as well as on the rest of the Task-features.

4.1.2. Cumulative sum transformation

We are dealing with complex time-series of tasks influenced by factors such as non-normal distributions, serial correlations, and fluctuating averages and variances. Cumulative sums are widely recognized transformations that bolster the analysis of sequential data, empowering users to gain insights into the intricate characteristics embedded within



```
...
{"Event":"SparkListenerTaskStart",...,"Task ID":7,...,"Launch Time":1664651148340
...
{"Event":"SparkListenerTaskStart",...,"Task ID":8,...,"Launch Time":1664651148380
...
{"Event":"SparkListenerTaskEnd",...,"Task Type":"ResultTask",{"Task ID":7,...
    "Task Metrics":{
        "Executor Deserialize CPU Time":11583500,
        "Executor Run Time":509,
        "Executor CPU Time":192748900,
        "Result Size":1545,"JVM GC Time":8,
        ...
        "Shuffle Read Metrics":
            {"Remote Blocks Fetched":1,
            "Remote Bytes Read":180408,
            "Total Records Read":1233},
            ...
            "Shuffle Write Metrics":
                {"Shuffle Bytes Written":0, ...},
                "Input Metrics":{"Bytes Read":0,"Records Read":0},
                "Output Metrics":{"Bytes Written":732660,"Records Written":1233}}
...
```

Task feature/Time-series scale	Task ID 0	… … …	Task ID 7	Task ID 8	… … …	Task ID N-1
Executor Deserialize CPU Time	… … …	… … …	11583500	… … …	… … …	… … …
Executor Run Time	… … …	… … …	192748900	… … …	… … …	… … …
… … …	… … …	… … …	… … …	… … …	… … …	… … …
Shuffle Write: Bytes Written	… … …	… … …	732660	… … …	… … …	… … …

Fig. 4. A portion of a *SparkEventLog* file sheds light on the profiling of task metrics thoughtfully via an ordered sequence of events. It is adhering to the formulation presented in Eq. (3).

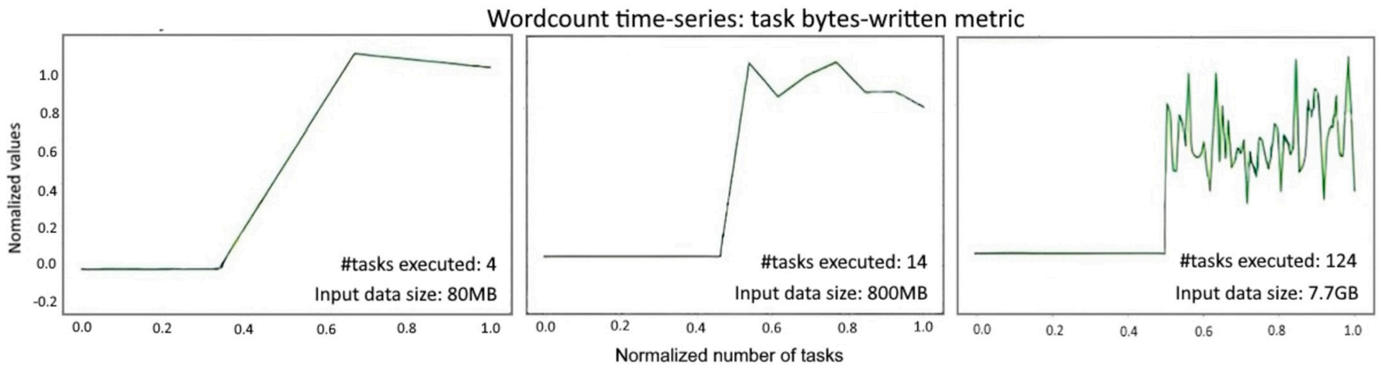


Fig. 5. Displayed time-series patterns of Wordcount for the task *bytes-written* metric across three input data sizes. The x-axis time scale, representing the number of tasks (IDs) executed in the workloads, has been normalized to facilitate visualization.

time-series data. In their publication [34], the authors introduced a robust statistical method for executing a specific transformation, hereinafter referred to as Cusum, on intricate sequential values. This method harmoniously aligns with the complexities of our time-series $\mathbb{TF}$.

In detail, the Cusum is the cumulative sum of standardized deviations from a Task-feature, calculated as a running sum of Tf normalized to its mean (μ) and standard deviation (σ). Hence, to calculate the Cusum, Tf is first standardized:

$$z_{m,n} = (tf'_{m,n} - \mu_{m,:})/\sigma_{m,:} \tag{6}$$

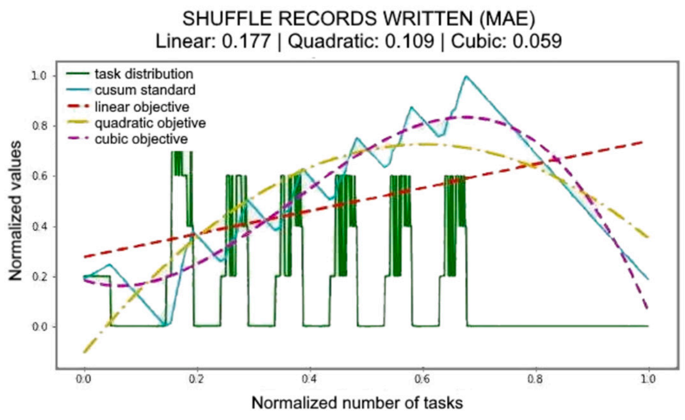
where $z_{m,n}$ is the standardized value for $tf_{m,n}$, the n^{th} value in Tf_m time-series. Second, the cumulative sum of standardized values is calculated:

$$z'_{m,n} = \begin{cases} z_{m,n} + z_{m,n-1}, & \text{if } n > 1 \\ z_{m,n}, & \text{if } n = 1 \end{cases} \tag{7}$$

The resulting Task-feature Cusum time-series (Tf_s) has a $\mu = 0$ and $\sigma = 1$, leading to z' values being multiples of σ. In Cusum space, positive z' values indicate a deviation of tf' above the mean, while negative values indicate a deviation below the mean. The slopes in the Cusum trends, whether decreasing or increasing, indicate that the values are (on average) below or above the Tf mean, respectively. This approach allows us to move towards our final workload descriptor, which is characterized by applying a higher-order polynomial regression.

4.1.3. Higher-order polynomial regression

To map the complete Cusum time-series into a latent space, we employ the well-established higher-order polynomial regression (PR) algorithm. This mapping process involves identifying the optimal polynomial fit of a specified degree that encapsulates the prevailing trends and patterns. Consequently, this strategy efficiently diminishes the impact of noise and minor fluctuations inherent in the time series. These attributes make it particularly well-suited for our objectives, as it provides interpretability and simplicity, in contrast to the utilization of



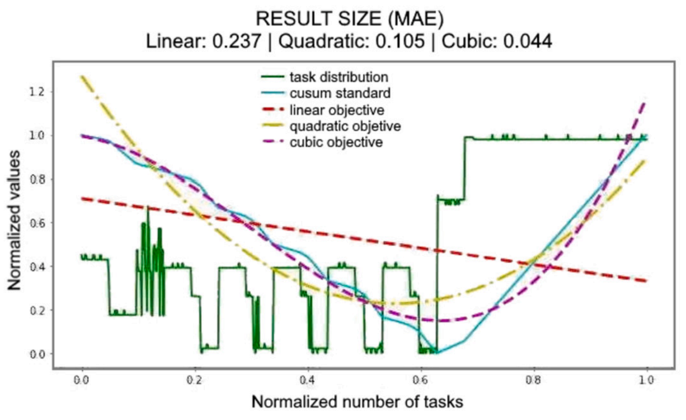


Fig. 6. E.g., Workload Task-features characterized by High-order polynomial regressions.

more intricate techniques such as neural networks. The coefficients of the approximating polynomial sequence are determined using the Least Squares Method, which minimizes the squared deviations of the approximating sequence from the experimental values of the time-series. To assess the accuracy of the regression, we utilize the Mean Absolute Error (MAE), which is a prevalent measure of forecast error in time-series analysis, described as follows:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{8}$$

where n represents the number of points, y_i is the actual observed value at time i, and $\hat{y}_i$ is the predicted or interpolated value at time i. The vertical bars $|\cdot|$ denote the absolute value.

Fig. 6 shows the different transformations of Task-features from the original time-series values, passing through the Cusum process, and finally fitted by linear, quadratic, and cubic regressions. These outcomes strongly indicate that a single linear trend falls short in representing the observed features adequately. Regressions that accommodate nonlinear behavior offer a notably more accurate approximation. For instance, cubic polynomial regressions generally exhibit a better fit for time-series based on the MAE. However, as a preview of our upcoming result analysis, our key observation is that the quadratic order provides a better trade-off between bias and variance for the purpose of our research.

Lastly, in accordance with the guidelines presented in Eq. (9a), each Tf_s is characterized using higher-order polynomial regression (Tf_p), with β denoting the coefficients and ξ representing the MAE. Therefore, all Tf_p are standardized to possess an equal count of features, providing a rationale for the comparison of diverse workloads. In the final stage of our characterization phase, we concatenate ($\bigoplus$) all Tf_p to obtain the definitive feature vector (as expressed in Eq. (9b)). This introduces a novel representation of an executed Spark workload (W_d) in latent space for downstream objectives.

$$\text{PR}(Tf_s) = \beta_0 + \beta_1 x^1 + \cdots + \beta_i x^i + \xi \;\Rightarrow\; Tf_p = \left[\beta_0,\, \beta_1, \cdots,\, \beta_i,\, \xi\right] \tag{9a}$$

$$W_{u=m+1,\, v=i+1+n} = \begin{bmatrix} \beta_{0,0} & \beta_{0,1} & \beta_{0,2} & \cdots & \beta_{0,i} & \xi_1 \\ \beta_{1,0} & \beta_{1,1} & \beta_{1,2} & \cdots & \beta_{1,i} & \xi_2 \\ \beta_{2,0} & \beta_{2,1} & \beta_{2,2} & \cdots & \beta_{2,i} & \xi_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \beta_{m,0} & \beta_{m,1} & \beta_{m,2} & \cdots & \beta_{m,i} & \xi_n \end{bmatrix} = \begin{bmatrix} Tf_{p_1} \\ Tf_{p_2} \\ Tf_{p_3} \\ \vdots \\ Tf_{p_n} \end{bmatrix} \Rightarrow \tag{9b}$$
$$W_{1,\, u\cdot v} = \left[Tf_{p_1} \oplus Tf_{p_2} \oplus Tf_{p_3} \oplus, \cdots, \oplus Tf_{p_n}\right] = W_d$$

4.2. Similar pattern recognition phase

As a downstream objective, we expand our framework to recognize similarities among workloads. We define similarity as the degree of resemblance between workloads, whether they are of the same type or not, based on their descriptors characteristics. This provides insights for enhancing Spark-based solutions, including optimization considerations.

To address similarity pattern recognition, we employ unsupervised machine learning techniques with a dual objective. First, detecting clusters or groups of workloads that exhibit similar behaviors but may not be readily discernible within the workload descriptors. The effectiveness of this goal will be evaluated based on the quality and meaningfulness of the patterns uncovered based on the type of workloads. Second, selecting relevant features that contribute significantly to better pattern discrimination, enhancing the efficiency of the downstream objective. Besides, it contributes to reduction in dimensionality avoiding irrelevant characteristics, while retaining essential information. The successful achievement of these objectives ensures that our framework not only uncovers latent patterns but also provides a clear and interpretable representation of workload similarities.

4.2.1. Clustering algorithms

The goal of clustering is to identify inherent groupings within a set of unlabeled data, where objects within each group are similar according to a certain criterion. Clustering algorithms can be divided into five categories: partitioning, hierarchical, density-based, grid-based, and model-based methods. In this similar pattern recognition phase, we prioritize the application of partitioning (K-Means) and hierarchical (Agglomerative) algorithms to a set $\mathbb{W}$ (Eq. (1)) of workload descriptors (Eq. (9b)), which have been previously standard min-max scaling to [0, 1] range. These representative algorithms are widely used and well-suited for identifying meaningful patterns or structures within the data. Furthermore, in preliminary analysis, they exhibited the best performance based on the requirements of this study.

In our proposal, the K-Means partitional algorithm uses the Euclidean distance to determine the proximity between the workload descriptors and the corresponding centroids to form the cluster. Besides, we rely on an improvement of the K-Means (K-Means++) algorithm proposed in [7]. It introduced an improved initialization step in K-Means which selects centroids that are farther apart, reducing the likelihood to converge to a suboptimal solution. Conversely, the Agglomerative clustering is a "bottom-up" approach where each W_d is assigned to an individual cluster at the initial step of the observation. Then, the clusters are progressively merged until they become one cluster. We define the Agglomerative clustering by applying a Ward's linkage method [19]. It links clusters based on the same method as the K-Means (Euclidean distance) to merge existing clusters.

4.2.2. Feature selection

Unsupervised feature selection (UFS) methods can be categorized, according to the strategy used for selecting features, mainly as filter, wrapper, and hybrid methods [38] as in supervised feature selection methods [15,9]. Focusing on UFS methods based on the filter approach,

Table 2
Showing the details of 216 samples with 24 unique HiBench workloads, including information on input data sizes and the maximum number of executed tasks. The (*) symbol indicates that the SQL plan has been generated by the Spark optimizer for the workload. The values within parentheses indicate the number of tasks created by Spark.

Workloads		Input data sizes		
Category	Type	Small	Medium	Large
Micro	Sleep	0 (4)	0 (9)	0 (9)
	Sort	9.4 KB (202)	824.0 KB (202)	85.0 MB (202)
	Terasort	6.4 MB (202)	640.0 MB (206)	6.4 GB (248)
	WordCount	9.4 KB (4)	82.1 MB (4)	821.2 MB (14)
	Repartition	716.8 KB (202)	716.8 MB (206)	7.1 GB (248)
SQL	Aggregation*	0 (200)	0 (200)	0 (200)
	Join*	40.3 KB (801)	572.9 KB (801)	5.3 MB (801)
	Scan*	0 (1)	0 (1)	0 (1)
Web Search	Page Rank	84.3 KB (16)	34.0 MB (24)	4.1 GB (24)
Machine Learning	Bayes*	107.3 MB (24)	127.8 MB (51)	440.9 MB (51)
	Kmeans*	25.5 MB (271)	6.7 GB (271)	46.0 GB(621)
	Logistic Regr.	21.9 MB (8260)	2.1 GB (8895)	275.0 GB (9581)
	ALS	381.4 KB (1402)	2.0 MB (1402)	18.8 MB (1402)
	PCA	51.0 KB (476)	262.1 KB (436)	16.3 MB (434)
	GBT	926.1 KB (29300)	36.6 MB (65300)	2.9 GB (138020)
	Random Forest	165.0 KB (1725)	2.8 MB (2605)	51.1 MB (2601)
	SVD	852.9 KB (224)	16.1 MB (220)	64.0 MB (220)
	Linear Regr.*	330.4 KB (23396)	1.2TB (23189)	3.6TB (42795)
	LDA*	251.8 MB (2394)	1.1 GB (2358)	2.8 GB (2355)
	SVM	501.1 MB (22904)	50.1 GB (22904)	4.0TB (45204)
	GMM*	22.1 MB (247)	4.0 GB (232)	26.0 GB (392)
	Correlation*	1.2 GB (635)	2.2 GB (438)	3.3 GB (635)
	Summarizer	40.2 MB (213)	16.0 GB (219)	30.8 GB (430)
Graph	NWeight	271.9 MB (232)	2.1 GB (232)	4.8 GB (232)

these can be categorized as univariate and multivariate. Within the univariate filter methods, two main groups can be highlighted: methods that assess the relevancy of each feature based on Information Theory, and those methods that evaluate features based on Spectral Analysis (manifold learning) using the similarities among objects. Regarding the last group, SPEC (SPECtrum decomposition) [41] is a univariate filter that evaluates the relevance of a feature by its consistency with the structure of the graph induced from the similarities among objects.

On the other hand, NDFS (Nonnegative Discriminative Feature Selection) [28] is a good performance and interpretability multivariate filter method. It is based on Spectral Analysis derived from the SPEC based on Spectral Analysis combined with Sparse Learning. Sparse Learning refers to those methods that seek a trade-off between some goodness-of-fit measure and the sparsity of the results. In order to enhance this similar pattern recognition phase, we rely on SPEC/NDFS univariate/multivariate filters because they are one of the most referenced and relevant UFS. These filter methods provide a feature ranking as output, making it possible to obtain a sorted list of the most relevant features of the workload descriptors within $\mathbb{W}$.

5. Experimental design

This section outlines the details of the experimental setup and the metrics used to evaluate the effectiveness of our proposed framework. Moreover, we conduct a compatible comparative analysis with an established workload characterization approach in the literature.

5.1. Experimental setup

We bypass the necessity of a large Spark infrastructure since our experiments prioritize capturing workload-generated patterns over emphasizing execution time. Notably, current trends in Spark cluster deployment lean towards container-based methods [18]. Hence, our experiments were carried out within a containerized environment [4]. We set up a Spark cluster on a computer with 36 GB RAM and 16 vCores, consisting of 1 master and 3 worker nodes. The cluster is based on Ubuntu 20.04, Spark v2.4.8, and Hadoop v2.7.7 images, which include Hive and Hadoop Distributed File System (HDFS). Workloads are launched into Spark cluster using the properly parameterized *spark-submit* command.

To assess our framework, we integrate all 24 Spark task metrics (Table 1). We also introduce an additional derived task metric (*tasks per stage*), adhering to the same principles, resulting in a total of 25 Task-features. Furthermore, we have selected all 24 available Spark workloads from HiBench, detailed in Table 2. These selections were made considering their diversity across application domains and internal attributes, including the existence of built-in SQL plans and the nature of tasks involved. Input data sizes (small, medium, and large) are associated to workload-specific configurations in Hibench, leading to non-uniform sizes across workloads. For instance, Page Rank uses the specific parameter *'hibench.pagerank.pages'*, which represents its number and does not accurately reflect the size in bytes. Hence, we present the total input bytes linked to the workloads to improve clarity.

The SQL workloads, defined by a zero input data size, are tailored to exclusively engage with Hive data, thus abstaining from accessing data stored in HDFS. Notably, this last constraint is applicable to the Sleep workload as well. We executed each workload on the three different input data sizes to assess its scalability and ability to handle datasets of varying sizes. Plausible Spark application settings regarding the cluster setup and input data sizes under which each workload runs are shown in Table 3. These settings are based on the most relevant properties described in Section 2 that have a significant impact on performance [31]. This approach allows us to comprehensively assess the effectiveness and robustness of our framework.

Specifically, we conducted an comprehensive analysis on a set of $216 = 24$ (unique workloads) $\cdot$ 3 (settings) $\cdot$ 3 (input data sizes) relevant workload descriptors. This set, hereafter denoted as $\mathbb{W}$ as per Eq. (2), was thoughtfully selected to ensure the reliability of the results, even with a limited number of samples, yet sufficient for validating and comparing our proposal. By carefully curating high-quality samples, our

Table 3
Experimental design: Workload samples across various plausible application settings and input data sizes. The number of shuffle partitions applies exclusively to workloads that involve Spark-SQL transformations (Table 2).

Setting id	Executor instances	Executor cores	Executor memory	Shuffle partitions	Applying to datasets	Total samples
1	1	4	4 GB	200	small - medium	48
2	3	3	3 GB	200	small - medium - large	72
3	3	3	3 GB	100	small - medium - large	72
4	3	3	5 GB	200	large	24

aim is to filter out extraneous factors, such as inappropriate application settings, thereby avoiding the generation of irrelevant workload descriptors. This approach underscores our commitment to deriving meaningful insights, where each workload descriptor contributes significantly to the overall understanding of the problem.

It is worth noting that each workload was executed multiple times to ensure the consistency of the built feature descriptor. To validate this, we computed the Euclidean distance between runs. Our analysis showed that the differences were negligible for our purposes, thus we chose one arbitrary run of each sample. To prevent Spark from dynamically creating and managing executors, we disabled *'dynamic allocation'* property in our experiments. Any remaining parameters that were not set explicitly were left at their default values. Ultimately, given that our analysis relies on distance-based measurements, we interchangeably use the term "points" to refer to the samples or "data points" to denote $\mathbb{W}$.

5.2. Workload descriptors evaluation metric

To geometrically evaluate the variability of the workload descriptors (W_d) across the proposed application settings, we calculate the mean Euclidean distance (μED) between descriptors, considering its corresponding input data sizes (ids). The calculation is performed as follows:

$$\mu\text{ED}_{(W_d,\,ids)} = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \|x_i - x_j\|_2 \tag{10}$$

where, x_i and x_j represent the i^{th} and j^{th} points, and $\|\cdot\|_2$ represents the Euclidean distance. The double summation computes the sum of the distances between every pair of distinct points in $\mathbb{W}$. The final value is then normalized by the total number of pairs of points ($n(n-1)$) to obtain the mean distance between the points. To establish a foundation for distance-based comparisons, we utilize the concept of the furthest point pair (FPP), as detailed in Equation (11). The FPP is defined as the greatest Euclidean distance encountered between any two points within $\mathbb{W}$, akin to conceptualizing the diameter ($\varnothing$):

$$\varnothing\text{FPP}_{\mathbb{W}} = \max_{i,j} \|x_i - x_j\|_2 \tag{11}$$

In effectively harnessing geometric comparisons, we must address the *curse of dimensionality* [26] phenomenon, where distances between points in high-dimensional spaces converge to a constant. Consequently, as dimensions increase, data sparsity emerges, impacting clustering accuracy and thereby reducing interpretability and modeling effectiveness. To address this phenomenon, we utilize Nearest Neighbor Density (NND) [20], which quantifies the average distance to k-nearest neighbors for each point, and becomes relatively invariant in higher dimensions. By leveraging NND, we determine the influence of the *curse of dimensionality*, calculated as:

$$\text{NND}(x_i) = \frac{1}{k} \sum_{j \in \text{NearestNeighbors}(x_i)} \|x_i - x_j\|_2 \tag{12}$$

where NND(x_i) quantifies the average distance of x_i to its k nearest neighbors, reflecting the density of neighboring points around x_i. The parameter k specifies the number of nearest neighbors considered, and NearestNeighbors(x_i) represents the set of k points that are closest to x_i based on Euclidean distance.

5.3. Clustering evaluation metrics

In the field of clustering evaluation, determining the effectiveness and efficiency of an algorithm can present a significant challenge. As a result, a wide range of evaluation methods have been developed to assess the performance of clustering algorithms [10]. These methods allow us to objectively measure the quality of clustering results, providing us with a deeper understanding of the strengths and limitations of different algorithms. Generally speaking, clustering validation techniques can be classified into two categories [35]: internal (based on the information intrinsic to the data alone) and external (based on previous knowledge about data) metrics. Although we are facing real problems which do not provide prior information about the workloads, external evaluation metrics are useful for allowing us to compare the results obtained from unsupervised models against a ground truth class assignment. Indeed, these metrics function as tools for evaluating the accuracy of the similarity pattern recognition within our framework.

5.3.1. Internal metrics

Based on the information intrinsic to the data alone, a clustering can be considered to be good when it has a high separation (inter-cluster) between clusters and a high compactness (intra-cluster) within clusters. Rather than addressing these aspects individually, we lean on an internal metric that endeavors to quantify both measures within a singular score [29]:

- Silhouette score: It measures the difference between the distance from a point of a cluster to other points of the same cluster and the distance from the same point to all the points of the closest cluster, defined as follows:

$$\text{Silhouette} = \frac{1}{n} \sum_{i=1}^{n} \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{13}$$

where n is the number of points, $a(i)$ represents the average distance of points i to other points within the same cluster (cohesion), and $b(i)$ is the smallest average distance of point i to points in different clusters (separation). The score is bounded between -1 for incorrect clustering and $+1$ when clusters are highly dense and well separated. Scores around zero indicate overlapping clusters.

5.3.2. External metrics

Given the knowledge of the ground truth class for the points, external metrics enable the measurement of the similarity with the predicted clusters. The most considerable external evaluation metrics [6,32] are based on matching sets and information theory. Set matching metrics assume that there is a one-to-one mapping between clusters and classes. Information theory-based metrics are a category of clustering evaluation metrics that quantify the similarity between the cluster assignments based on the concept of information entropy. Anticipating upcoming equation explanations, we use C to denote the set of predicted class clusters being evaluated relative to their corresponding ground truth classes, denoted as G. The number of distinct classes is symbolized by m, and n represents the total point count in the set.

- F-Measure is calculated as the harmonic mean of Purity and Inverse Purity, which both belong to the matching sets class. Purity measures the degree to which clusters are composed of points from the same class, i.e., it penalizes the noise in a cluster. However, it does not reward grouping points from the same class together. Purity is computed by taking the weighted average of maximal precision values.

$$\text{Purity}(C,G) = \frac{1}{n}\sum_{i=1}^{k}\max_{j=1}^{m}\frac{n_{ij}}{n_i}, \tag{14}$$

where n_{ij} represents the number of points that belong to both the ith cluster and the jth class, n_i represents the size of the ith cluster. Inverse Purity rewards grouping points together, but it does not penalize mixing points from different classes. It focuses on the cluster exhibiting the highest representation for each class, defined as:

$$\text{Inverse Purity}(C,G) = \frac{1}{n}\sum_{j=1}^{m}\max_{i=1}^{k}\frac{n_{ij}}{n_j}, \tag{15}$$

where n_j represents the size of the jth class. The remaining parameters maintain consistency with the definitions in the previous equation.
Let's now elucidate the F-Measure, a more robust metric matching each class with the cluster that has a highest combined Purity and Inverse Purity, as follows:

$$\text{F-Measure} = \frac{(1+\beta^2)\times\text{Purity}\times\text{Inverse Purity}}{\beta^2\times\text{Purity}+\text{Inverse Purity}} \tag{16}$$

where the parameter β controls the trade-off between Purity and Inverse Purity. We set the value of β to 1, which is commonly used. All set matching metrics are bound by from 0 to 1, where the higher value the better.
- Normalized Mutual Information (NMI) is categorized within the information theory-based. NMI measures the degree of similarity between the predicted class clusters and the true classes, taking into account the sizes of the clusters and the classes. NMI ranges from 0 to 1, where a value of 1 indicates perfect agreement between two labelings, and a value of 0 indicates no agreement beyond chance. Consistent with the earlier notations, it is expressed as:

$$\text{NMI} = \frac{2I(C,G)}{H(C)+H(G)}, \tag{17}$$

where $I(C,G)$ is the Mutual Information (MI) between the two labelings:

$$\text{I}(C,G) = \sum_{i=1}^{k}\sum_{j=1}^{m}\frac{n_{ij}}{n}\log\frac{n_{ij}n}{n_i n_j}, \tag{18}$$

and $H(C)$ and $H(G)$ are the entropies of the predicted and ground truth classes, respectively:

$$\text{H}(C) = -\sum_{i=1}^{k}\frac{n_i}{n}\log\frac{n_i}{n};\quad \text{H}(G) = -\sum_{j=1}^{m}\frac{n_j}{n}\log\frac{n_j}{n} \tag{19}$$

By analyzing all of the above metrics, we obtain a comprehensive evaluation of the clustering algorithms applied to the workload descriptor points. In the context of similarity pattern recognition, we place particular emphasis on evaluating the external metrics, which are commonly used and well-suited for this purpose. It is important to note that for all points, ground truth classes were removed prior to conducting clustering and feature selection analysis. This was done to ensure that our similarity pattern recognition framework was able to perform unsupervised algorithms, without relying on any prior knowledge of the class labels. In our utilization of clustering algorithms, we configure the number of clusters (k) to precisely match the distinct workload classes (i.e., 24), applying them comprehensively across the data points [37].

Last but not least, we conduct a compatible comparative analysis of our framework with the established workload characterization proposed by Prats et al. [33], which we refer to as the "base-approach". Guided by their methodology, we compute the mean, minimum, maximum, and standard deviation for each task metric. This results in workload descriptors of 100 dimensions $=$ 25 (task metrics) $\cdot$ 4 (statistical summaries) in $\mathbb{W}$, which we analyze following the similar pattern recognition methodology as our framework.

6. Results

In this section, we comprehensively validate the proposed framework by analyzing the results of the experimental design. Initially, we explore the impact of the Cusum transformations and higher-order polynomial regressions on the original time-series distributions. Next, we analyze how various application settings affect each workload descriptor, taking into account different input data sizes, from a geometric perspective. Subsequently, we assess the clustering algorithm results using both internal and external metrics to determine the grouping of workload descriptors into clusters and compare them with the base-approach. We also explore the advantages of unsupervised feature selection by ranking the most valuable attributes to enhance similarity pattern recognition performance.

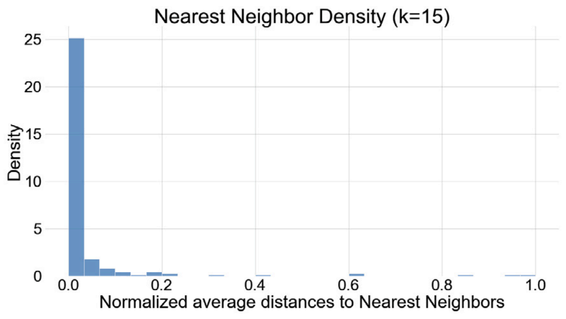
6.1. Exploring the impact of applying Cusum and higher-order polynomial regressions

The performance of different polynomial regressions on $\mathbb{W}$ is presented in Table 4, which offers valuable insights. In terms of the application of transformations, we found two primary observations. The first point is that without applying the Cusum transformation, the internal Silhouette score performs best in the first order for both algorithms. However, its performance worsens with the increase in the order of polynomial regressions. This behavior is also reflected in the external metrics. As we described in Section 4.1.3, the 1st order polynomial regression presents the worst MAE in characterizing the time-series of Task-features, despite suggesting the best density and separation of clusters. Nevertheless, this characterization does not adequately capture the workload patterns, leading to suboptimal performance in the external metrics. The second point is that by applying the Cusum transformation, we observed a decline in the internal metric but a significant improvement in external metrics, specifically in F-Measure when using a quadratic order polynomial regression with K-Means. Thereby, the Cusum transformation provides us with a capable mechanism to better capture the behavior of the application tasks, favoring similar pattern recognition of the workloads.

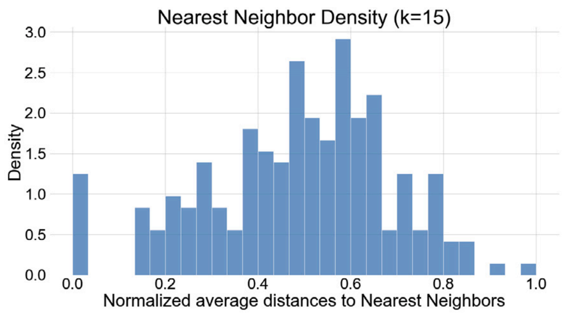
On the other hand, although cubic order polynomial regressions generally provide a better fit (lower MAE) for the workload descriptors, they exhibit the worst performance for both clustering algorithms. Achieving a better trade-off between bias and variance is crucial when selecting the order of the polynomial regression for characterizing the time-series. Higher-order models tend to reduce bias but can become too sensitive to specific patterns and increase variance, while lower-order models may oversimplify the data and increase bias. Thus, quadratic polynomial regression offered us a more reliable approximation based on the observed metrics, achieving better ability to generalize well to new workloads. These results highlight the importance of carefully considering the selection of the polynomial order, as higher-order do not necessarily lead to better clustering performance in our scenario. Nevertheless, the order can be understood as a hyperparameter of the workload characterization process (even separately for each Tf_s), which enables fine-grained tuning according to the feature-based downstream objectives.

Table 4
Cluster analysis results showing the impact of applying Cusum transformations and high-order polynomial regressions on all points of workload descriptors. The best metrics for each clustering algorithm are highlighted.

Clustering algorithm	Transform.	Order PR	Internal	External metrics	
			Density & Separation	Set Matching	Information Theory
			Silhouette	F-Measure	NMI
K-Means	None	1º	**0.418**	0.797	0.877
		2º	0.338	0.777	0.862
		3º	0.287	0.689	0.800
	Cusum	1º	0.368	0.809	0.874
		2º	0.321	**0.846**	**0.902**
		3º	0.257	0.729	0.808
Agglomerative	None	1º	**0.429**	0.807	0.891
		2º	0.374	0.762	0.866
		3º	0.301	0.701	0.819
	Cusum	1º	0.389	0.821	0.892
		2º	0.360	**0.836**	**0.902**
		3º	0.276	0.742	0.829



(a) The base-approach.



(b) Our approach.

Fig. 7. Nearest Neighbor Density of the workload descriptors in $\mathbb{W}$.

As we proceed with our result analysis, we adopt the Cusum and quadratic polynomial degree transformations as the standard approach. This leads to a $100 = 25$ (Task-features) $\cdot$ 4 (regression coefficients + error) dimensional vector, which defines a workload descriptor (Sec. 4.1.3).

6.2. *Analyzing workload descriptors from a geometric perspective*

In this section, we analyze the effect of the *curse of dimensionality* phenomenon. Then, we explore the geometric variability of workload descriptors under different approaches. In closing, we conduct a geometric study encompassing the application settings on all input data sizes simultaneously.

6.2.1. *The curse of dimensionality phenomenon*

To apply the geometric evaluation approach outlined in Section 5.2, we commence by evaluating the impact of the *curse of dimensionality*. This assessment involves analyzing the explained Nearest Neighbor Density (NND). We employ $k = 15$ for NND, a value commonly chosen as the square root of the set of data (thus, $\sqrt{216} \approx 15$). We conduct this analysis by comparing our workload characterization against the base-approach in the 100-dimensional space of $\mathbb{W}$, showed in the Fig. 7. In high-dimensional spaces affected by the *curse of dimensionality*, we expect a histogram of average Nearest-Neighbor distances to peak at a certain constant value, reflecting the uniformity of distances [20]. Conversely, lower densities in the histogram suggest dispersion in the space, which is indeed a useful characteristic for the similar pattern recognition.

As depicted in Fig. 7a, the base-approach shows a pronounced peak, indicating a high density of points at a constant average distance. This reveals a strong convergence in the vicinity, complicating the clustering of the workload descriptors. Contrastingly, our approach, as shown in Fig. 7b, exhibits a histogram characterized by significant variations in average distances and lower densities. This indicates that workload descriptors are widely dispersed across the 100-dimensional space. In consequence, this observation suggests that the *curse of dimensionality* might have a limited impact on our approach, highlighting its resilience. However, the base-approach appears to be more susceptible to the phenomenon. Armed with these insights, we are well-prepared to advance to the subsequent stages of our analysis.

6.2.2. *Exploring geometrically the variability of workload descriptors*

In order to facilitate the geometric exploration of our workload descriptors, we proceeded to calculate the ⌀FPP (diameter) of $\mathbb{W}$, yielding a value of 27.55. The diameter provides a valuable measure into the distribution and dispersion of the workload descriptors within $\mathbb{W}$ (akin to conceptualizing the shape), thereby enhancing our understanding of their spatial arrangement and variability.

To assess the pairwise distances between workload descriptors for each input data size and their relationship to the diameter, we employ the diameter-to-mean ratio criterion. This ratio is computed by dividing the Euclidean mean distances of the workload descriptors by the diameter, as outlined in Eq. (20). In our study, we employ a 0.35 ratio (35% of the diameter) as a proximity threshold. This ratio strikes a balance between covering a diameter segment and avoiding overly conservative or lenient values, while effectively capturing meaningful distance variations. A lower ratio indicates a more closer set of workload descriptors,

Table 5
Mean Euclidean distances of workload descriptors, encompassing all three application settings for each Spark workload, are computed separately for the three distinct input data sizes (ids), as well as for all together. Ratios exceeding the threshold are indicated with an asterisk (*). The values within parentheses represent the calculated ratios obtained using Eq. (20).

Workload	$\mu\text{ED}_{(W_d,ids)}$ (ratio)			
	Small	Medium	Large	All
Sleep	3.69 (0.13)	6.73 (0.24)	6.30 (0.22)	5.74 (0.20)
Sort	7.63 (0.27)	7.34 (0.26)	5.21 (0.18)	7.42 (0.26)
Terasort	4.95 (0.17)	4.97 (0.18)	7.63 (0.27)	11.5 (0.41)*
WordCount	5.23 (0.18)	4.49 (0.16)	2.27 (0.08)	6.89 (0.25)
Repartition	1.81 (0.06)	4.77 (0.17)	8.78 (0.32)	10.8 (0.39)*
Aggregation	2.89 (0.10)	3.86 (0.14)	3.38 (0.12)	3.32 (0.12)
Join	6.30 (0.22)	8.04 (0.29)	8.12 (0.29)	7.75 (0.28)
Scan	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
Page Rank	3.73 (0.13)	6.49 (0.23)	4.89 (0.17)	6.57 (0.23)
Bayes	5.62 (0.20)	8.07 (0.29)	5.01 (0.18)	6.56 (0.23)
Kmeans	7.20 (0.26)	8.63 (0.31)	6.29 (0.22)	9.84 (0.35)
LR	4.24 (0.15)	6.10 (0.22)	6.03 (0.21)	6.96 (0.25)
ALS	9.08 (0.32)	6.10 (0.22)	5.96 (0.21)	9.03 (0.32)
PCA	4.25 (0.15)	5.49 (0.19)	3.06 (0.11)	10.0 (0.36)*
GBT	7.51 (0.27)	7.70 (0.27)	7.21 (0.26)	8.89 (0.32)
RF	8.24 (0.29)	4.48 (0.16)	1.44 (0.05)	9.37 (0.34)
SVD	4.47 (0.16)	6.71 (0.24)	10.0 (0.36)*	8.14 (0.29)
LM	3.27 (0.11)	6.68 (0.24)	2.61 (0.09)	4.43 (0.16)
LDA	6.32 (0.22)	3.69 (0.16)	4.28 (0.15)	5.73 (0.20)
SVM	4.68 (0.16)	3.63 (0.13)	3.99 (0.14)	6.55 (0.23)
GMM	5.87 (0.21)	5.84 (0.21)	10.0 (0.36)*	9.83 (0.35)
Correlation	3.69 (0.13)	6.82 (0.24)	4.97 (0.18)	10.6 (0.38)*
Summarizer	7.44 (0.27)	8.47 (0.30)	8.50 (0.30)	8.62 (0.31)
NWeight	7.21 (0.26)	4.56 (0.16)	3.38 (0.12)	4.99 (0.18)
Mean ratios	(0.171)	(0.208)	(0.191)	(0.276)

while a higher ratio suggests a more spread-out set, within a range of [0,1]. Table 5 provides the computed ratios and average Euclidean distances for reference.

$$\text{ratio} = \frac{\mu\text{ED}_{(W_d,ids)}}{\varnothing\text{FPP}_{\mathbb{W}}} \tag{20}$$

The findings of our analysis demonstrate that different combinations of settings can lead to significant variations in the descriptors, highlighting the substantial impact of setting choices on the Spark application [5]. When examining the ratios segmented by input data sizes, we observed that the majority of ratios fell below the threshold. However, there were a few exceptions, such as SVD and GMM on the big data size, which exhibited slightly higher values. When considering the all input data sizes, we also noticed that certain specific workloads, such as Terasort, Repartition, Correlation, and PCA, exhibited relatively sparse characteristics. Despite their sparsity, these specific workloads still maintained a reasonable distance ratio, as it was computed across all input data sizes collectively. Scan workloads deserve special mention for showing a minimum distance, which can be attributed to their unchanging internal execution behavior. They only execute commands to Hive data storage, resulting in a stable pattern that remains minimal across the different settings.

Our analysis confirms that our workload descriptors effectively capture the variations introduced by the different settings applied to the Spark application. It is worth noting that the consistency of the variability is evident as the descriptors exhibit proximity with respect to the analyzed workload types. This observation is further supported by mean ratios consistently falling below the threshold. Additionally, the diameter can function as a mechanism for detecting data drift when new workload descriptors are introduced. Our motivation is to compare the distance of new workload descriptors to the existing set (or centroids of the clusters) against the diameter (or ratio). This approach enables us to identify any significant deviation or shift in the data distribution, which could indicate data drift.

Table 6
Individually presenting the results of applying clustering algorithms to each segmented input data size, as well for all input data sizes together. Scores outside parentheses belong to our framework, while scores within parentheses denote results from the (base-approach). We highlight the best results that align with our goals, as determined by external metrics.

Clustering algorithm	Input data size	Internal	External metrics	
		Density & Separation	Set Matching	Information Theory
		Silhouette	F-Measure	NMI
K-Means	small	0.398 (0.537)	0.916 (0.725)	0.955 (0.834)
	medium	0.370 (0.423)	0.916 (0.755)	0.955 (0.845)
	large	0.542 (0.611)	0.965 (0.870)	0.983 (0.928)
	all	0.321 (0.437)	**0.846** (0.558)	**0.902** (0.677)
Agglomerative	small	0.422 (0.550)	0.937 (0.728)	0.967 (0.839)
	medium	0.383 (0.454)	0.951 (0.777)	0.975 (0.868)
	large	0.542 (0.615)	0.965 (0.872)	0.983 (0.930)
	all	0.360 (0.446)	0.836 (0.578)	0.902 (0.695)

The demonstrated capabilities of our framework in constructing generic and robust workload descriptors can be extended to other distributed computing systems like Apache Flink [3], which similarly focuses on tasks as core units of computation. Thus, it could allow profiling of their metrics to create workload descriptors using the proposed approach.

6.3. *Similar pattern recognition*

Defining the characteristics that determine similarity can be a challenging task as it can be based on multiple criteria, which must be carefully selected and evaluated. For example in terms of internal Spark behavior, workloads that involve similar computation and data processing patterns can be considered similar. Thus, the workloads such as Sort, Terasort, and Wordcount involve similar computation characteristics such as sorting and counting, while workloads such as K-Means, GBT, and SVM involve machine learning computations. Similarly, workloads such as Pagerank, LDA, and SVD involve graph processing and matrix computations.

On the other hand, in terms of DAG operations, the workloads can be categorized based on the type and amount of data shuffle, stages, and tasks involved. For example, Sort, Terasort, and Join involve a large amount of data shuffle, while sleep and scan have no shuffle. Tree aggregation is mainly used in decision tree-based algorithms, such as GBT and Random Forest. Summarizer involves mostly narrow transformations and a small amount of shuffle. Exchange is used in Repartition to move the data across the nodes in the cluster. In the next section, we assess the performance of our framework in recognizing workload patterns using their descriptors, as well as comparing it with the base-approach.

6.3.1. *Exploring the impact of input data sizes by clustering algorithms*

Upon analyzing the clustering metric results in Table 6, we observe that the base-approach exhibits better performance in terms of the internal metric Silhouette. However, our framework consistently outperforms it in terms of external metrics, which prioritize robustness in recognizing the same type of workloads across different application settings and input data sizes. This demonstrates that our approach excels at capturing meaningful patterns within the workload descriptors, aligning with our primary objective.

Exploring the results of our framework, we observe accurate pattern recognition, especially when considering individual input data sizes and Agglomerative clustering. Notably, identical outcomes are observed

Table 7
Results of unsupervised feature selection based on ranking applied to all points. Scores outside parentheses belong to our framework, while scores within parentheses denote results from the (base-approach). We highlight the results that best align with our goals.

Clustering	UFS	#Features	Silhouette	F-Measure	NMI
K-Means	Baselines	100	0.321 (0.437)	0.846 (0.558)	0.902 (0.677)
	SPEC	97 (80)	0.332 (0.393)	0.863 (0.627)	0.907 (0.726)
	NDFS	78 (17)	0.372 (0.544)	**0.909** (0.626)	**0.945** (0.740)
Agglomerative	Baselines	100	0.360 (0.446)	0.836 (0.578)	0.902 (0.695)
	SPEC	75 (83)	0.309 (0.454)	0.860 (0.593)	0.912 (0.708)
	NDFS	47 (16)	0.452 (0.560)	0.868 (0.653)	0.930 (0.765)

for specific external metrics, specifically with K-Means for small and medium input data sizes, as well as with both algorithms for large input data sizes. This consistency in outcomes can be attributed to the linear structures present in the workload descriptors. These structures emerge from transformations during the characterization phase, providing valuable assistance to the clustering algorithms.

Conversely, when encompassing all input data sizes (including the three different settings for each workload), K-Means displays a minor advantage in external metrics, particularly highlighting the F-Measure. It is worth noting that the decrease in overall performance on both algorithms is a logical behavior as the workload descriptors are strongly related to their settings and input data sizes. This is desirable, as it enables the discovery of new clusters, which can help in identifying more consistent workload similar patterns.

6.3.2. *Exploring the impact of clustering by applying unsupervised feature selection*

Our study is enriched by applying UFS to $\mathbb{W}$, which selects a concise subset of informative features. This allows for a comprehensive comparison of outcomes with the baselines, representing the best scores achieved thus far for both our framework and the base-approach across all features. UFS aims to enhance the accuracy and interpretability of the clustering algorithms. The results are summarized in Table 7.

While both UFS techniques performed better than the baselines, our main findings indicate that NDFS significantly outperformed SPEC. In particular, with NDFS, Agglomerative clustering produced denser and better separated clusters compared to the baselines in terms of internal evaluation metrics. Analyzing the base-approach, an general improvement is achieved by applying Agglomerative clustering. However, it consistently exhibits poor accuracy on external metrics, indicating limitations in its ability to extract meaningful using only 16 features. Turning attention to our framework, it demonstrates significantly improved performance in external metrics compared to the base-approach. This underscores our approach emphasis on robustness in recognizing similar workload patterns across various application settings and input data sizes. Notably, K-Means displayed superior F-Measure and NMI values, serving as the benchmark quality indicators for this study.

The additional insights presented in Table 8 offer a more comprehensive depiction of the clustering intricacies across all workloads generated by our framework. According to the experimental setup outlined in Section 5.1, an optimal clustering is achieved with an unique workload of (9). It ensures that all samples of the same workload are grouped together in a single cluster, without being merged with other clusters or workloads.

When examining the comparisons of clustering outcomes, we unveil a blend of both consistent and divergent assignments between the two algorithms. Approximately 42% of clusters perfectly converge in their assignments, particularly, clusters 1 to 10 highlight optimal assignments, emphasizing a mutual recognition of distinct workloads. Conversely, the remaining 58% of clusters exhibit some variations in their interpretations. Nevertheless, despite these variations, the broader clustering patterns maintain a remarkable alignment. Turning our attention to K-Means, it is noted that optimal clustering is evident in up to 58% of the clusters. Furthermore, by considering the inclusion of clusters 15, 16, 17, and 22 as near-optimal assignments, the overall percentage could rise to 75%.

Table 8
Comparison of clustering outcomes for workloads (counter samples) with unsupervised feature selection using NDFS. Workload names are presented in lowercase for improved readability.

Cluster	K-Means (#78-Dimension)	Agglomerative (#47-Dimension)
1	pagerank(9)	pagerank(9)
2	aggregation(9)	aggregation(9)
3	als(9)	als(9)
4	lda(9)	lda(9)
5	join(9)	join(9)
6	nweight(9)	nweight(9)
7	linear(9)	linear(9)
8	gbt(9)	gbt(9)
9	bayes(9)	bayes(9)
10	kmeans(9)	kmeans(9)
11	sleep(9)	sleep(9), scan(9)
12	scan(9)	wordcount(3)
13	gmm(9)	gmm(6)
14	wordcount(9)	wordcount(6)
15	svm(8)	svm(9)
16	rf(8)	rf(8)
17	correlation(7)	correlation(9), pca(6)
18	pca(6)	gmm(3)
19	pca(3)	pca(3), rf(1)
20	correlation(2)	sort(6)
21	svd(9), summarizer(9)	svd(9), summarizer(9)
22	lr(9), smv(1)	lr(9)
23	terasort(3)	terasor(3), repartion(3)
24	sort(9), terasort(6), repartion(6)	sort(3), terasort(6), repartition(6)

Interestingly, the Sort and Repartition workloads were clustered together, even though they did not appear to share similarities in terms of Spark's internal behavior. However, these workloads exhibited similar shuffling operations, which explained their clustering. In contrast, the SVD and Summarizer workloads, which differed in task shuffling behavior, were clustered together based on the DAG representation of tasks, such as executor CPU and executor deserialize. These results highlight that our framework is capable of identifying new patterns (hidden) that go beyond the well-known characteristics.

In conclusion, K-Means achieved the best external clustering metrics using NDFS on a subset of 78 features, implying that the majority of the full set of 100 features hold significance. This finding suggests that the workload descriptors encompass valuable, non-spurious features. The utilization of UFS techniques has proven effective in improving accuracy and enhancing the interpretability, while also mitigating the potential *curse of dimensionality* phenomenon. However, selecting the appropriate latent space dimension (subset of features) and clustering algorithm become important hyperparameters of the framework, involving trade-offs between dimensionality reduction, information retention, and computational cost.

7. Assessing practical applicability of the framework

In this section, we examine the real-world applicability of our framework, including a detailed analysis of algorithmic complexity. For com-

Table 9
Algorithmic complexities of the framework. Workload characterization phase[1]. Similar pattern recognition phase[2]. Validation results[3].

Algorithm	Time Complexity	Space Complexity
Cumulative Sum Transformation[1]	$O(n)$	$O(1)$
Higher-Order Polynomial Regression[1]	$O(n \cdot m^d)$	$O(m^d)$
K-Means Clustering[2]	$O(t \cdot n \cdot k \cdot m)$	$O(n \cdot k)$
Agglomerative Clustering[2]	$O(n^2 \cdot m)$	$O(n \cdot m)$
Nonnegative Discriminative Feature Selection[2]	$O(n \cdot m \cdot t)$	$O(n \cdot m)$
Furthest Point Pair[2,3]	$O(n^2)$	$O(1)$

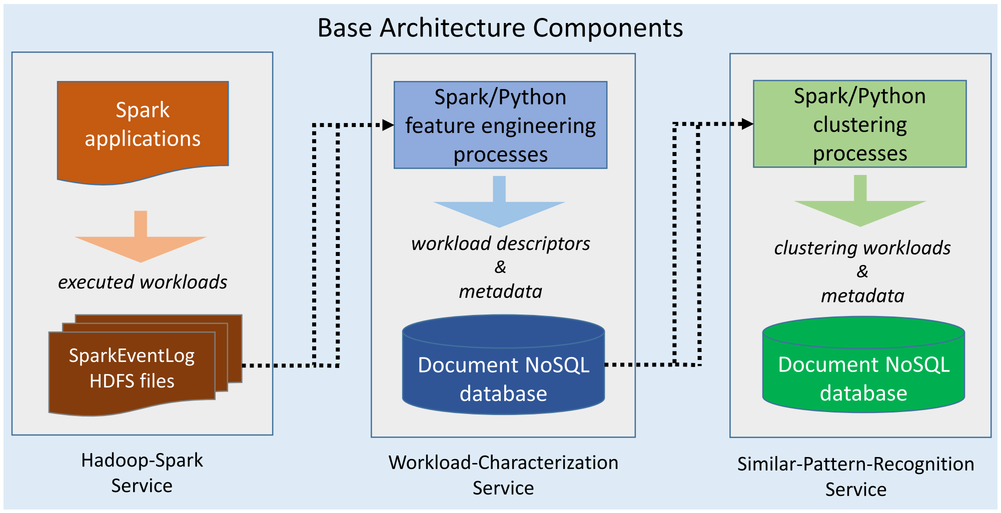

Base Architecture Components
Spark applications
executed workloads
SparkEventLog HDFS files
Hadoop-Spark Service
Spark/Python feature engineering processes
workload descriptors & metadata
Document NoSQL database
Workload-Characterization Service
Spark/Python clustering processes
clustering workloads & metadata
Document NoSQL database
Similar-Pattern-Recognition Service

Fig. 8. Architecture foundations for implementing our framework.

pleteness, we also present our foundational implementation architecture.

7.1. Analyzing computational algorithmic complexity

We carefully analyze the complexity of the algorithms within our framework, revealing the resources they require in real-world usage. We provide a condensed summary of their complexities in Table 9. Time complexity relates to number of samples (n), while space complexity gauges memory use during execution.

Detailed insights for each algorithm are outlined below:

- Cumulative Sum Transformation: This algorithm exhibits linear time complexity in relation to n due to its uncomplicated aggregation nature, resulting in minimal memory usage.
- Higher-Order Polynomial Regression: The time complexity is influenced by the degree (d) of the polynomial, the dimension of features (m), and n. Note that the computational complexity escalates rapidly with higher d and an increased m, due to the greater number of computations and terms involved in fitting a higher-degree polynomial. Correspondingly, the space complexity depends on both d and m, reflecting the necessity to store intermediate matrix computations.
- K-Means Clustering: The time complexity is determined by various parameters, including the number of iterations (t), centroids (k), n, and m. The space complexity scales with the number of centroids (k) and n.
- Agglomerative Clustering: Its quadratic time complexity concerning n and m emphasizes its sensitivity to n. This characteristic renders it especially well-suited for a reasonable scale of n.
- Nonnegative Discriminative Feature Selection: The time complexity is closely linked to n, m, and the maximum number of iterations t. The space complexity aligns with n and m. Despite this, the computational demands of the algorithm remain manageable across varying dimensions of n.
- Furthest Point Pair: The time complexity is $O(n^2)$ brute force algorithm. It requires $O(1)$ space to store the largest distance found so far and the two points that are farthest apart. Another approach is the divide-and-conquer algorithm, which has a time complexity of $O(n \log n)$.

It is crucial to emphasize that real-world performance of these algorithms is influenced by factors beyond theoretical complexities. Hardware capabilities, efficient algorithm implementation, and adept use of parallelization strategies significantly impact practical outcomes. Our framework excels in this context with strong potential for logical parallelization during workload characterization. This is evident in concurrent computation of related algorithms for each Spark workload. Additionally, parallelizing individual Task-feature characterization can enhance the time-intensive polynomial regressions. Furthermore, specialized languages (e.g., PySpark) can boost algorithm performance, aligning seamlessly with parallelization opportunities in modern real-world systems.

Finally, when compared to the base approach, our framework is more computationally demanding due to the inclusion of additional algorithms. Despite this, our framework remains practical in real-world scenarios, thanks to moderate computational requirements for most algorithms. The use of lower-order polynomials, especially in time-consuming regressions, contributes to this practical feasibility.

7.2. Architecture foundations

We present the base architecture of our framework, which includes three well-defined components (services) as depicted in Fig. 8.

Our primary focus lies in understanding the nuances of information exchange among these services:

- Spark-Hadoop Service: Every executed application generates a event log file identified by a unique identifier, which is stored in HDFS.
- Workload Characterization Service: Converts Spark event log files into nested complex JSON documents, encompassing workload identification and metadata. This metadata includes application settings, workload descriptor vectors, and various intermediate transformations, such as original time-series tasks, statistical aggregations, cumulative sums, and polynomial regressions. This information enriches workload interpretability and bolsters experimentation, all stored within a document-oriented database.
- Similar Pattern Recognition Service: Identifies clustering executions for tracing purposes. Clustering workload outcomes, such as centroids, ratios, diameters, and cluster labels, are stored in a document-oriented database as metadata for further analysis

8. Conclusion

This paper presents a novel framework for Spark workload characterization and pattern recognition capabilities. Our framework distinguishes itself by relying solely on quantitative metrics at the application task-level for workload characterization. The approach employs non-intrusive techniques to individually characterize workloads, achieved by diversely transforming task metrics. The first phase of the framework culminates in the creation of robust vector descriptors, capturing the essence of executed Spark workloads, tailored for representation within a latent space. Within this space, we incorporate consistent measurements from a geometric perspective, adding a layer of meaningful interpretation. As part of capabilities for expansion, our workload characterization process effectively handles new workloads and incorporates new task metrics, including those introduced by new Apache Spark versions. In the second phase, our framework is extended to recognize patterns by integrating unsupervised machine learning techniques for clustering and feature selection. The experiments showcase its effectiveness in identifying similar workloads, achieving high accuracy as indicated by F-Measure (90.9%) and NMI (94.5%). These results strongly outperform the base-approach comparison, which scored an F-Measure of 57.8% and an NMI of 69.5%. For completeness, we explore the real-world applicability of our framework, including a detailed analysis of algorithmic complexity and the foundational implementation architecture.

In light of all the above, this research can hold relevance for other distributed computing systems, as well as offer insights for optimizing Spark-based solutions. Our future research endeavors will be focused on expanding the framework to encompass auto-tuning for big data workloads.

Abbreviations

ALS	Alternating Least Square
GBT	Gradient Boosted Trees
GMM	Gaussian Mixture Modeling
LDA	Linear Discriminant Analysis
LM	Linear Model (linear regression)
LR	Logistic Regression
RF	Random Forest
SQL	Structured Query Language
SVD	Singular Value Decomposition
SVM	Support Vector Machine

Funding

This work was supported by CITIC, as Research Center accredited by Galician University System, which is funded by "Consellería de Cultura, Educación e Universidade from Xunta de Galicia", supported in an 80% through ERDF Funds, ERDF Operational Programme Galicia 2014-2020, and the remaining 20% by "Secretaría Xeral de Universidades" (Grant ED431G 2019/01). It was also partially funded by Xunta de Galicia/FEDER-UE under Grant ED431C 2022/44; Ministerio de Ciencia e Innovación MCIN/AEI/10.13039/501100011033 under Grant PID2019-109238 GB-C22.

CRediT authorship contribution statement

Mariano Garralda-Barrio: Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Writing – original draft. **Carlos Eiras-Franco:** Supervision, Writing – review & editing. **Verónica Bolón-Canedo:** Project administration, Supervision, Writing – review & editing.

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

Executed workload logs used in this study are available to download: SparkEventLogs.

Acknowledgments

The authors would like to express their gratitude to Delivery Networks by Minsait (Indra company), for providing the infrastructure used in this study.

References

[1] Apache Spark - Unified Engine for large-scale data analytics, https://spark.apache.org/, 2018.
[2] Monitoring and instrumentation spark, https://spark.apache.org/docs/2.4.8/monitoring.html, 2018.
[3] Apache Flink, https://flink.apache.org/, 2018.
[4] Docker images for apache spark executed on hadoop yarn, https://github.com/mgarralda/hadoop-spark-cluster, 2021.
[5] N. Ahmed, A.L. Barczak, T. Susnjak, M.A. Rashid, A comprehensive performance analysis of apache hadoop and apache spark for large scale data sets using hibench, J. Big Data 7 (1) (2020) 1–18, https://doi.org/10.1186/s40537-020-00388-5.
[6] E. Amigó, J. Gonzalo, J. Artiles, F. Verdejo, A comparison of extrinsic clustering evaluation metrics based on formal constraints, Inf. Retr. 12 (4) (2009) 461–486, https://doi.org/10.1007/s10791-008-9066-8.
[7] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, Tech. Rep., Stanford, 2006.
[8] A.J. Awan, M. Brorsson, V. Vlassov, E. Ayguade, Micro-architectural characterization of apache spark on batch and stream processing workloads, in: 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), 2016, pp. 59–66.
[9] V. Bolón-Canedo, N. Sánchez-Maroño, A. Alonso-Betanzos, Feature selection for high-dimensional data, Prog. Artif. Intell. 5 (2016) 65–75, https://doi.org/10.1007/978-3-319-21858-8.
[10] M. Brun, C. Sima, J. Hua, J. Lowey, B. Carroll, E. Suh, E.R. Dougherty, Model-based evaluation of clustering validation measures, Pattern Recognit. 40 (3) (2007) 807–824, https://doi.org/10.1016/j.patcog.2006.06.026.
[11] G. Cheng, S. Ying, B. Wang, Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model, J. Syst. Softw. 180 (2021) 111028, https://doi.org/10.1016/j.jss.2021.111028.
[12] G. Cheng, S. Ying, B. Wang, Y. Li, Efficient performance prediction for apache spark, J. Parallel Distrib. Comput. 149 (2021) 40–51, https://doi.org/10.1016/j.jpdc.2020.10.010.
[13] T. Chiba, T. Onodera, Workload characterization and optimization of tpc-h queries on apache spark, in: 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE, 2016, pp. 112–121.
[14] S. Chidambaram, S. Saraswati, R. Ramachandra, J.B. Huttanagoudar, N. Hema, R. Roopalakshmi, Jvm characterization framework for workload generated as per machine learning benchmark and spark framework, in: 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), IEEE, 2016, pp. 1598–1602.

[15] C. Eiras-Franco, V. Bolón-Canedo, S. Ramos, J. González-Domínguez, A. Alonso-Betanzos, J. Tourino, Multithreaded and spark parallelization of feature selection filters, J. Comput. Sci. 17 (2016) 609–619, https://doi.org/10.1016/j.jocs.2016.07.002.
[16] M. Genkin, F. Dehne, Autonomic workload change classification and prediction for big data workloads, in: 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 2835–2844.
[17] M. Genkin, F. Dehne, P. Navarro, S. Zhou, Machine-learning based spark and hadoop workload classification using container performance patterns, in: C. Zheng, J. Zhan (Eds.), Benchmarking, Measuring, and Optimizing, Springer International Publishing, Cham, ISBN 978-3-030-32813-9, 2019, pp. 118–130.
[18] T. Gunawardena, K. Jayasena, Real-time uber data analysis of popular uber locations in kubernetes environment, in: 2020 5th International Conference on Information Technology Research (ICITR), 2020, pp. 1–6.
[19] 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, https://doi.org/10.3390/s20154111.
[20] A. Hinneburg, C.C. Aggarwal, D.A. Keim, What is the nearest neighbor in high dimensional spaces?, in: Proc. of the 26th Internat. Conference on Very Large Databases, Cairo, Egypt, 2000, 2000, pp. 506–515, http://nbn-resolving.de/urn:nbn:de:bsz:352-opus-70224.
[21] S. Huang, J. Huang, J. Dai, T. Xie, B. Huang, The hibench benchmark suite: characterization of the mapreduce-based data analysis, in: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010), vol. 2, 2010, pp. 41–51.
[22] M. Janecek, N. Ezzati-Jivan, S.V. Azhari, Container workload characterization through host system tracing, in: 2021 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2021, pp. 9–19.
[23] Z. Jia, J. Zhan, L. Wang, R. Han, S.A. McKee, Q. Yang, C. Luo, J. Li, Characterizing and subsetting big data workloads, in: 2014 IEEE International Symposium on Workload Characterization (IISWC), 2014, pp. 191–201.
[24] L. John, P. Vasudevan, J. Sabarinathan, Workload characterization: motivation, goals and methodology, in: Workload Characterization: Methodology and Case Studies. Based on the First Workshop on Workload Characterization, 1998, pp. 3–14.
[25] S. Jokar Jandaghi, A. Bhattacharyya, C. Amza, Phase annotated learning for apache spark: workload recognition and characterization, in: 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2018, pp. 9–16.
[26] E. Keogh, A. Mueen, Curse of dimensionality, in: Encyclopedia of Machine Learning and Data Mining, 2017, pp. 314–315.
[27] M. Lattuada, E. Gianniti, M. Hosseini, D. Ardagna, A. Maros, F. Murai, A. Couta da Silva, J.M. Almeida, et al., Gray-box models for performance assessment of spark applications, in: Proceedings of the 9th International Conference on Cloud Computing and Services Science, SciTePress, 2019, pp. 609–618.
[28] Z. Li, Y. Yang, J. Liu, X. Zhou, H. Lu, Unsupervised feature selection using nonnegative spectral analysis, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 26, 2012, pp. 1026–1032.
[29] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu, Understanding of internal clustering validation measures, in: 2010 IEEE International Conference on Data Mining, IEEE, 2010, pp. 911–916.
[30] S. Mustafa, I. Elghandour, M.A. Ismail, A machine learning approach for predicting execution time of spark jobs, Alex. Eng. J. 57 (4) (2018) 3767–3778, https://doi.org/10.1016/j.aej.2018.03.006.
[31] N. Nguyen, M.M.H. Khan, Y. Albayram, K. Wang, Understanding the influence of configuration settings: an execution model-driven framework for apache spark platform, in: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), 2017, pp. 802–807.
[32] J.-O. Palacio-Niño, F. Berzal, Evaluation metrics for unsupervised learning algorithms, arXiv preprint, arXiv:1905.05667, 2019, https://doi.org/10.48550/arXiv.1905.05667.
[33] D.B. Prats, F.A. Portella, C.H.A. Costa, J.L. Berral, You only run once: spark autotuning from a single run, IEEE Trans. Netw. Serv. Manag. 17 (4) (2020) 2039–2051, https://doi.org/10.1109/TNSM.2020.3034824.
[34] P. Regier, H. Briceño, J.N. Boyer, Analyzing and comparing complex environmental time series using a cumulative sums approach, MethodsX 6 (2019) 779–787, https://doi.org/10.1016/j.mex.2019.03.014.
[35] E. Rendón, I. Abundez, A. Arizmendi, E.M. Quiroz, Internal versus external cluster validation indexes, Int. J. Comput. Commun. Control 5 (1) (2011) 27–34, http://universitypress.org.uk/journals/cc/20-463.pdf.
[36] S. Shah, Y. Amannejad, D. Krishnamurthy, M. Wang, Quick execution time predictions for spark applications, in: 2019 15th International Conference on Network and Service Management (CNSM), 2019, pp. 1–9.
[37] S. Solorio-Fernández, J.F. Martínez-Trinidad, J.A. Carrasco-Ochoa, A new unsupervised spectral feature selection method for mixed data: a filter approach, Pattern Recognit. 72 (2017) 314–326, https://doi.org/10.1016/j.patcog.2017.07.020.
[38] S. Solorio-Fernández, J.A. Carrasco-Ochoa, J.F. Martínez-Trinidad, A review of unsupervised feature selection methods, Artif. Intell. Rev. 53 (2) (2020) 907–948, https://doi.org/10.1007/s10462-019-09682-y.
[39] W. Xiong, Z. Yu, Z. Bei, J. Zhao, F. Zhang, Y. Zou, X. Bai, Y. Li, C. Xu, A characterization of big data benchmarks, in: 2013 IEEE International Conference on Big Data, 2013, pp. 118–125.
[40] Z. Yu, W. Xiong, L. Eeckhout, Z. Bei, A. Mendelson, C. Xu, Mia: Metric importance analysis for big data workload characterization, IEEE Trans. Parallel Distrib. Syst. 29 (6) (2018) 1371–1384, https://doi.org/10.1109/TPDS.2017.2758781.
[41] Z. Zhao, H. Liu, Spectral feature selection for supervised and unsupervised learning, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 1151–1157.

Mariano Garralda-Barrio holds a Technical Engineering degree in Computer Systems from UNED in Spain. He obtained an official M.S. degree in Informatics Engineering with a specialization in big data analytics from the University of Lleida. He completed an official M.S. degree in Artificial Intelligence Research with a focus on learning and data science from UIMP in Spain, in collaboration with AEPIA. Currently, he is pursuing a Ph.D. degree in computational science at the University of A Coruña. He works as a Senior Practice Manager in big data and AI engineering at Indra company.

Carlos Eiras-Franco received his Ph.D. from Universidade da Coruña in 2020 where he is currently an Assistant Professor. His research spans anomaly detection systems, predictive maintenance and explainable artificial intelligence, among other machine learning problems, with a focus on scalability. He completed research stays at the University College of London (United Kingdom), Universidade do Minho (Portugal), and Technische Universität Berlin (Germany) and has taken part as a researcher in more than 30 R&D projects, both publicly and privately funded. In the course of his academic career, he has written numerous research papers in high impact scientific journals and he has participated in several international conferences.

Verónica Bolón-Canedo received her B.S. (2009), M.S. (2010) and Ph.D. (2014) degrees in Computer Science from the Universidade da Coruña (Spain). After a postdoctoral fellowship in the University of Manchester, UK (2015), she is currently an Associate Professor in the Department of Computer Science of the Universidade da Coruña. Her main current research areas are machine learning and feature selection. She is co- author of more than 100 papers on these topics in international conferences and journals.