Orchestrating Big Data Analysis Workflows in the Cloud: Research Challenges, Survey, and Future Directions

MUTAZ BARIKA and SAURABH GARG, University of Tasmania
ALBERT Y. ZOMAYA, University of Sydney
LIZHE WANG, China University of Geoscience (Wuhan)
AAD VAN MOORSEL, Newcastle University
RAJIV RANJAN, China University of Geoscience (Wuhan) and Newcastle University

Interest in processing big data has increased rapidly to gain insights that can transform businesses, government policies, and research outcomes. This has led to advancement in communication, programming, and processing technologies, including cloud computing services and technologies such as Hadoop, Spark, and Storm. This trend also affects the needs of analytical applications, which are no longer monolithic but composed of several individual analytical steps running in the form of a workflow. These big data workflows are vastly different in nature from traditional workflows. Researchers are currently facing the challenge of how to orchestrate and manage the execution of such workflows. In this article, we discuss in detail orchestration requirements of these workflows as well as the challenges in achieving these requirements. We also survey current trends and research that supports orchestration of big data workflows and identify open research challenges to guide future developments in this area.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Computer systems organization → Cloud computing**; • **Computing methodologies → Distributed algorithms**; • **Computer systems organization → Real-time systems**;

Additional Key Words and Phrases: Big data, cloud computing, workflow orchestration, research taxonomy, approaches, and techniques

ACM Reference format:
Mutaz Barika, Saurabh Garg, Albert Y. Zomaya, Lizhe Wang, Aad van Moorsel, and Rajiv Ranjan. 2019. Orchestrating Big Data Analysis Workflows in the Cloud: Research Challenges, Survey, and Future Directions. *ACM Comput. Surv.* 52, 5, Article 95 (September 2019), 41 pages.
https://doi.org/10.1145/3332301

This research is supported by an Australian Government Research Training Program (RTP) Scholarship. This research is also partially funded by two Natural Environment Research Council (UK) projects including (LANDSLIP:NE/P000681/1 and FloodPrep:NE/P017134/1).
Authors' addresses: M. Barika and S. Garg, Discipline of ICT | School of Technology, Environments and Design (TED), College of Sciences and Engineering, University of Tasmania, Hobart, Tasmania, Australia 7001; emails: {mutaz.barika, saurabh.garg}@utas.edu.au; A. Zomaya, School of Computer Science, Faculty of Engineering, J12 - Computer Science Building. The University of Sydney, Sydney, New South Wales, Australia; email: albert.zomaya@sydney.edu.au; L. Wang, School of Computer Science, China University of Geoscience, No. 388 Lumo Road, Wuhan, P. R China; email: lzwang@cug.edu.cn; A Van Moorsel, School of Computing, Newcastle University, Newcastle upon Tyne, NE1 7RU, United Kingdom; email: aad.vanmoorsel@ncl.ac.uk; R. Ranjan, School of Computer Science, China University of Geoscience, No. 388 Lumo Road, Wuhan, P. R China, and School of Computing, Newcastle University, Newcastle upon Tyne, NE1 7RU, United Kingdom; email: raj.ranjan@ncl.ac.uk.

0360-0300/2019/09-ART95 $15.00
https://doi.org/10.1145/3332301

1 INTRODUCTION

In recent years, Big Data has gained considerable attention from governments, academia, and enterprises. The term "big data" refers to collecting, analyzing, and processing voluminous data. Those interested in big data are looking for ways to efficiently store and analyze their large datasets to distill useful information [22]. However, it is difficult for traditional data processing platforms to process large datasets with a great variety of types. Similarly, traditional data processing applications that relied on these platforms are incapable of achieving the intended analytical insights to make better decisions. Hence, many big data platforms have recently been proposed for transacting with big data, facilitating the design and building of big data analysis applications to ingest, and processing as well as analyzing tremendous volumes of data.

The complexity of supporting big data analysis is considerably larger than the perception created by recent publicity. Unlike software solutions that are specifically developed for some application, big data analytics solutions typically require the integration of existing trusted software components in order to execute the necessary analytical tasks. These solutions need to support the high velocity, volume, and variety of big data (i.e., 3Vs of big data [78]) and thus should leverage the capabilities of cloud datacenter computation as well as storage resources as much as possible. In particular, many of the current big data analytics solutions can be classified as data-driven workflows, which integrate big data analytical activities in a workflow. Analytical tasks within these big data workflow applications may require different big data platforms (e.g., Apache Hadoop or Storm) as well as a large amount of computational and storage resources to process large volume and high velocity data. Intrusion detection, disaster management, and bioinformatics applications are some examples of such applications.

Big data workflows are very different from traditional business and scientific workflows (see Appendix B), as they have to continuously process heterogeneous data (batch and streaming data) and support multiple active analytical tasks at each moment in time. Moreover, they involve analytical activities that have heterogeneous platform and infrastructure requirements, and the overall workflows can be highly dynamic in nature because processing requirements at each step are determined by data flow dependencies (the data produced in earlier steps in the workflow) as well as control flow dependencies (the structural orchestrations of data analysis steps in the workflow). In addition, big data workflows are different from streaming operator graphs that formed by streaming processing systems like Apache Storm and Flink, as they have heterogeneous analytical activities, and involve multiple data sources that inject their data into upstream or/and downstream analytical activities and multiple outputs; but these systems employ continuous operator models to process streaming data only and have one cluster, and they form operator graphs with one feeding data source and one sink operator.

The focus of previous workflow taxonomy studies [1, 46, 102, 141] are either on business processes and information systems (for G. M. Giaglis [46]) or Grid computing and its applications (for J. Yu and R. Buyya [141] and Rahman et al. [102]). Given the advancement in big data applications and systems, new surveys are required that can synthesize current research and help in directing future research. Some recent surveys (such as by Sakr et al. [109, 110] and Mansouri et al. [87]) focused on a specific aspect of the big data applications and their management. They have not given overall dimensions involved in their orchestration such as workflow initialization and parallelization. For example, Sakr et al. [110] focused on big data analysis with MapReduce model research, while Sakr et al. [109] and Mansouri et al. [87] studied only data management aspects for deploying data-intensive applications in the cloud. Recently, Liu et al. [79] provided a survey covering scheduling frameworks for various big data systems and a taxonomy based on their features without any indepth analysis of issues related to big data analytical workflows and their orchestration.

However, big data workflow applications processing big data exhibit different patterns and performance requirements that traditional workflow processing methods and current workflow management systems cannot handle efficiently. Therefore, we require research into new orchestration models as well as orchestration platform and management technologies that can provide services to support the design of big data workflows, the selection of resources (including at platform and infrastructure), and the scheduling and deployment of workflows. These needs for future research drive us to investigate the answer of the following research questions in this article: (1) what are the different models and requirements of big data workflow applications?, (2) what are the challenges based on the nature of this type of workflow application and cloud + edge datacenters that we will face when developing a new big data orchestration system?, and (3) what are the current approaches, techniques, tools and technologies to address these challenges?

To assist in this future aim and answer the above research questions, we identify the requirements of big data workflow orchestration systems for management of such workflows execution on the cloud. We discuss the current state of the art, provide research taxonomy, and list open issues. The article makes the following concrete research contributions:

— An exhaustive survey of big data programming models (see Section 3.3). We further elaborate on this survey to explain the relationship between big data programming models and workflows (Appendix A).
— Propose a comprehensive research taxonomy to allow effective exploration, assessment, and comparison of various big data workflow orchestration issues (see Section 5) across multiple levels (workflow, data, and cloud).
— Apply the proposed research taxonomy for surveying (see Section 6) a set of carefully chosen big data workflow orchestration tools (see Appendix C), orchestration techniques, and research prototypes.
— Identify current open research issues (see Section 8) in the management of big data workflows based on the literature survey and requirements.

This article is structured as follows: Section 2 compares the proposed research taxonomy against the literature. Section 3 presents a typical example of big data workflow that spans the three layers (workflow, data, and cloud) and its orchestration in a cloud system. Section 4 highlights the key important requirements of big data workflows while in Section 5, we present a taxonomy for challenges in fulfilling those requirements. Section 6 presents the current approaches and techniques to address these challenges. Section 7 reviews scientific workflow systems with data-intensive capabilities and big data orchestrating systems, and discusses the capabilities of big data orchestrating systems against the presented research taxonomy. Section 8 presents and discusses the open issues for further research, while Section 9 concludes the article.

2 RELATED WORK: POSITIONING VERSUS EXISTING TAXONOMIES

Several previous studies [1, 46, 102, 141] focus on understanding either business process modeling or scientific workflow orchestration tools and techniques. As discussed earlier, big data workflow applications present a unique set of requirements that require a fresh perspective on researching orchestration tools and techniques.

In a narrow context, Liu et al. [79] provided a survey and taxonomy of existing big data workflow orchestrators (e.g., Yet Another Resource Negotiator (YARN), Mesos), focusing on scheduling techniques. Sakr et al. [110] provided a survey of MapReduce-based approaches and techniques for the development of large-scale data analysis solutions. In another paper, Sakr et al. [109] presented a survey of big data storage solutions (e.g., HadoopDB, HyperTable, Dryad) for managing

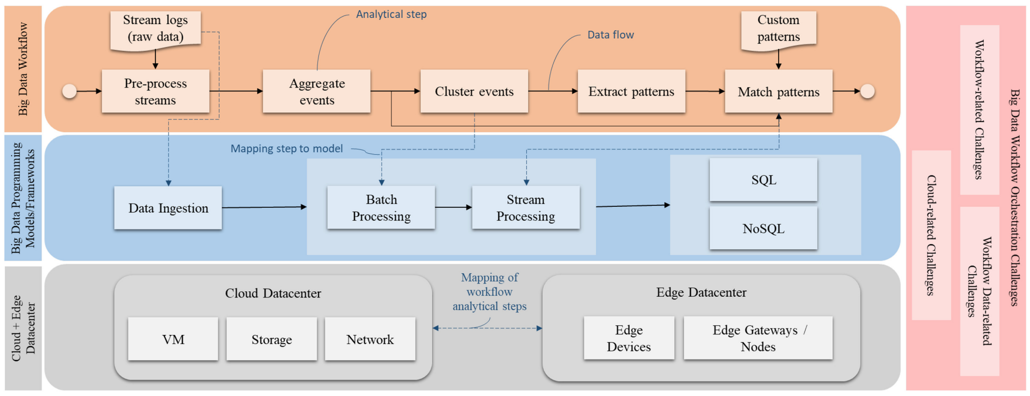

Fig. 1. An example workflow for anomaly detection over sensor data streams and its mapping to programming models/frameworks and cloud + edge datacenters.

big data in cloud environments. Similarly, Mansouri et al. [87] presented a taxonomy and survey of cloud-based big data storage management tools.

To the best of our knowledge, existing taxonomies and surveys are limited to one aspect of big data or data-intensive applications (e.g., MapReduce data processing model [110] or data management [87, 109]). In contrast, we present a holistic research taxonomy to understand end-to-end issues related to orchestrating big data workflow applications on clouds. We propose a taxonomy that gives exhaustive classification of big data workflow orchestration tools and techniques from the perspective of additional (sub-)dimensions contributing to future development by giving "in-depth" analysis of existing works. In other words, this work aims to alleviate the research gaps in understanding the big data workflows.

3 BIG DATA WORKFLOW ORCHESTRATION

3.1 Representative Example of a Big Data Workflow

To aid understanding of big data workflows and the issue of orchestrating such workflow applications in the cloud and edge resources, we present a typical example of anomaly detection (shown in Figure 1). It is a representation of the workflow presented in Ref. [2].

The data pipeline is used to analyze sensor data streams for online anomaly detection. The representation of this workflow spans the three layers (workflow, data, and cloud) is shown in Figure 1. First of all, the streams of data (i.e., stream logs) are ingested in the pipeline by following a message ingestion model (i.e., Kafka), where all events that are collected within a window of time are pre-processed by filtering and enriching them with additional metadata, e.g., external timestamps. Next, aggregation of events is done, for example, per region or sensor type in a given window of time, which get clustered into different categories and later passed to pattern matching step (last step). At the cluster events step, a clustering-based outlier detection algorithm will run in a batch fashion over all produced aggregated events in order to generate outliers (possible/proposed anomalies). After that, all outliers are mined to extract possible frequent patterns, and those extracted patterns are further transformed into complex event processing queries reliant on the selected strategy. Finally, all patterns are matched to output the outliers by constantly injecting the rules into distributed complex event processing engines, and these engines perform continuous queries on the streams of data coming from either the pre-processing step or aggregate step for online anomaly detection. Accordingly, the stream programming model is followed for processing and analyzing sensor data streams ingested in this workflow to produce continuous insights (online anomaly detection) by using Apache Storm [2]; also, the anomaly patterns and analysis results generated in this workflow could be stored in Structured Query Language (SQL) or NoSQL databases.

From the above example, we can easily see that the analytical tasks included in the data pipeline require seamless coordination for real-time and dynamic decision making, handling different types of heterogeneity and uncertainties such as changing in data velocity or data volume. That includes (1) fulfilling the need of diverse computational models for pre-processing streams, aggregating and clustering events, and extracting possible frequent patterns; (2) managing inter-dependent analytical tasks, where any change in execution and performance characteristics of one can affects the downstream steps; and (3) matching patterns' analytical tasks needed to take the advantage of edge resources available at edge datacenters to perform edge analytics, avoiding any possible latency. Therefore, to achieve this seamless execution for such types of workflow, various programming tasks need to be performed, leading to several challenges related to cloud + edge resources and data orchestration, which span over three different levels (workflow, data, and cloud).

3.2 Workflow Level

One of the aims of the big data workflow orchestration platform is to manage the sequence of analytical tasks (formed workflow application) that needs to deal with static as well as dynamic datasets generated by various data sources. This includes various programming tasks, i.e., workflow composition and workflow mapping [104]. Workflow composition is to combine different analytical tasks, where their workloads are dependent on each other and any change made in the execution and characteristics of one step affects the others. Therefore, different users of the workflow define their requirements and constraints from different contexts, resulting in different analytical tasks of a workflow needing to be executed, where the requirements are not only different but may also conflict with each other. Accordingly, a workflow orchestration system should provide the guidance for domain experts to define and manage the entire pipeline of analytical tasks, data flow, and control flow, and their Service Level Agreement (SLA) and Quality of Service (QoS) needs. It can support different workflow orchestration techniques to compose heterogeneous analytical tasks on cloud and edge resources including script-based (that defines composition flow using script languages), event-based (that uses event rules defined in workflow language to provide responsive orchestration process), or adaptive orchestration (that dynamically adopts composition flow in accordance to application and execution environment needs). Internet of Things (IoT) and Cyber-Physical Systems (CPS) have several requirements such as processing a large amount of data streams from the physical world, the real-time decision making to sensor events and dynamic management of data flow [112]. In other words, IoT and CPS applications are adaptive workflows involving event-driven tasks that sophistically analyse data streams for decision making. These workflows can be considered as a specific exemplar of big data pipeline and managed under the umbrella of big data workflow orchestration system and techniques, as big data workflows consist of dynamic and heterogeneous analytical activities, where data arrives in different formats, at different volumes, and at different speeds [150].

Workflow mapping is to map the graph of analytical tasks to big data programming platforms (e.g., batch analytical task to Apache Hadoop, streaming analytical task to Apache Storm), cloud resources, and edge resources. It also needs to consider different configuration possibilities (configuration of each big data programming framework, e.g., number of map and reduce tasks with Apache Hadoop in the context of batch processing; configuration of cloud resources, e.g., the type of resource and the location of datacenter; configuration of edge resources, e.g., type of edge device and network latency), This requires a cross-layer resources configuration selection technique in the big data orchestration system to select custom configurations from plenty of possibilities.

As a result, several challenges have emerged due to the complexity and dynamism of big data workflow including workflow specification languages, initialization, parallelization and scheduling, fault-tolerance, and security. Since the heterogeneous and dynamic nature of cloud + edge resources bring additional challenges (we will discuss this at Cloud + Edge Datacenter level), these challenges further complicate those workflow-related challenges.

3.3 Big Data Programming Models/Frameworks Level

The processing of big data requires heterogeneous big data programming models, where each one of them provides a solution for one aspect. Within big data workflow, various computational models may be required for involved analytical tasks, where one analytical task may also need distinct computation models based on the characteristics of data (batch processing for static datasets, stream processing for dynamic datasets, hybrid processing for static and dynamic datasets). SQL and NoSQL models are also utilized for storing data to cope with volume and velocity of data. Therefore, understanding these models is essential in selecting the right big data processing framework for the type of data being processed and analyzed.

These different models cover the ingesting, storing, and processing of big data. The MapReduce programming model (batch-oriented model) and stream programming model are used for data processing, NoSQL/SQL models are used for data storage, and message ingestion models are used for data importing. In this section, we will review these models and compare them to outline the main differences.

The complex and dynamic configuration requirements of big data workflow ecosystems calls for the need to design and develop new orchestration platforms and techniques aimed at managing: (1) sequence of analytical activities (formed workflow application) that needs to deal with static as well as dynamic datasets generated by various data sources; (2) heterogeneous big data programming models; and (3) heterogeneous cloud resources.

3.3.1 MapReduce Programming Model. The MapReduce programming model [31] is a leading batch-oriented parallel data programming model that is intended for processing complex and massive volumes of data at once (static data) to gain insights. It was developed at Google Research and relied on the following functions: Map and Reduce. The input data (finite large datasets) is stored firstly in Hadoop Distributed File System (HDFS). Then, the input data is split into smaller chunks, and then these chunks are processed in a parallel and distributed manner by Map tasks that generate intermediate key-value pairs. After that, these pairs are aggregated by Reduce function. Due to the finiteness property, this model has the capability to perform computation on data in stages, where it can wait until one stage of computation is done before beginning another stage of computation, allowing it to perform jobs just as sorting all intermediate results globally [55]. Moreover, in respect of increasing the future computation load, this model allows us to scale horizontally by adding more workers to cope with such loads. This model exploits data locality to schedule computation tasks to avoid unnecessary data transmission [56].

3.3.2 Stream Programming Model. In this model, data arrives in streams, which are assumed to be infinite and are being processed and analyzed (in a parallel and distributed manner) as they arrive and as soon as possible to produce incremental results [55, 56]. The sources of streams could be, for example, mobile and smart devices, sensors, and social media. Thus, the stream computation in the stream programming model is assumed to process continuous incoming streams with low latency (i.e., seconds and minutes of delays), instead of processing a very large dataset in hours and more [75]. There are two approaches to achieve this kind of processing/computation. The native stream processing approach processes every event as it arrives in succession, resulting in the lowest-possible latency, which is considered as the advantage of this approach; nevertheless, the disadvantage of this approach is that it is computationally expensive because it processes every incoming event. The micro-batch processing approach aims to decrease the cost of computation for the processing stream by treating the stream as a sequence of smaller data batches; in this approach, the incoming events are divided into batches by either time of arrival or once batch size hits a certain threshold, resulting in the reduction of processing computational cost, but could also bring together more latency [69, 82]. With this model, stream computations are independent of each other, which means there is no dependency or relation among them. Moreover, in respect of increasing future computation load, this model allows us to scale vertically and horizontally to cope with such loads. Due to data-flow graphs implementing both data programming models, the stream-programming model can emulate batch processing. Apache Storm is one of the examples of the stream processing platform. In addition to stream-oriented big data platforms, a number of stream-oriented services are offered by various cloud providers, which deliver stream-oriented big data platforms as services. Examples of these services are Microsoft Azure Stream Analytics and IBM Streaming Analytics.

3.3.3 NoSQL/SQL Models. For storing big data, there are two models, which are: NoSQL model and SQL model. The NoSQL models (MongoDB, Amazon Dynamo, Cassandra, HyperTable, BigTable, HBase) provide access capabilities reliant on transactional programming primitives in which a specific key allows a search for a specific value. The use of these access primitives results in improving the scalability and predictions of performance, making it suitable for storing huge amounts of unstructured data (such as mobile, communication, and social media data). SQL data stores (Oracle, SQL Server, MySQL, PostGreSQL) organize and manage data in relational tables, where SQL as a generic language provides the ability to query as well as manipulate data. In essence, when transactional integrity (Atomicity, Consistency, Isolation, and Durability (ACID) properties) is a strict requirement, these data stores are more effective than NoSQL stores. However, both NoSQL and SQL data stores are likely to be used by future big data applications, and that is driven by data varieties and querying needs. SQL Models (Apache Hive, Apache Pig) provide the ability to query data over various cloud storage resources, e.g., Amazon S3 and HDFS, based on structured query language. In respect of increasing future load, the NoSQL model allows us to scale horizontally using sharding or partitioning techniques to cope with this future load; while the SQL model has limited capability to cope with such loads.

3.3.4 Message Ingestion Models. The message ingestion model is a publish-subscribe messaging pattern that allows us to import data from various sources and inject it as messages (i.e., events/streams) into big data platforms for processing to produce analytical insights, where the senders of messages are called publishers and the receivers of messages are called subscribers. The stream computations are independent to each other, which means there is no dependency or relation among them. Moreover, in respect of increasing future computation loads, this model can scale horizontally by adding more workers to cope with such loads. Relying on these models, message ingestion systems (such as Amazon Kinesis and Apache Kafka) achieve a durable, high-throughput, fault-tolerant, and low-latency queuing of streaming data.

Amazon Web Services (AWS) Kinesis is a cloud-based stream platform offered by Amazon. It provides powerful services that allow working easily with real-time streaming data (to load and analyze continuous data) in the AWS cloud, and the ability to develop and build custom streaming data applications to meet specific needs.

3.3.5 Hybrid Models. To support applications requiring both batch and stream data processing, hybrid models are developed. An example of a cloud service that implements hybrid data programming models (batch and stream) is Google cloud Dataflow. It is a Google fully-managed service for stream data processing and batch data processing as well. Dataflow is an unified execution framework and programming model for creating and executing both batch and stream pipelines to load, process, and analyze data, without having to pay attention to operational tasks such as resource management and performance optimization. As an execution framework, it handles the lifetime of resources transparently and provisions resources on demand to reduce latency while, at the same time, maintaining high utilization efficiency. Moreover, as a unified programming model, it uses the Apache Beam model that eliminates programming model switching cost between batch and streaming mode by providing the ability for developers to represent the requirements of computation without taking into consideration the data source.

Lambda Model—A batch-first approach uses batching for unifying batch and stream processing, where data streams are treated as micro-batches (collection of small batches). It supports batch processing for historical datasets to get insights according to the needs of users, and stream processing via micro-batching approach, which is suitable for applications where the data collection and availability through dashboards have time delays, and such data needs to be processed as it arrives [70]. Lambda model comprises three layers. Batch layer as a first layer is responsible for storing the master dataset and periodically precomputing the views of batch data. Speed layer as a second layer is responsible for processing online data as it is received in near real-time fashion to minimize latency. Serving layer as a third layer is responsible for consolidating both by combining the results from batch and speed layers to answering user queries. Lambda architecture achieves two properties of big data, which are velocity and volume. By using such architecture, users can determine which data parts need stream or batch processing in order to improve their data processing costs.

Kappa Model—A stream-first approach that considers all data as streams, whether such data is batch data or stream data. In contrast to Lambda architecture, this architecture, in favor of simplicity, dispenses the batch layer. Thus, there is no periodical recomputation for all data in the batch layer. Instead, the Kappa architecture performs all data computation in one system (i.e., stream processing system) and executes recomputation only when there is a change in business logic by rerunning historical data. This is accomplished by utilizing a powerful stream processor that is able to handle data at a higher rate than the incoming data rate as well as a scalable streaming system for data retention. Kappa architecture comprises two layers. The speed layer manages the processing of stream data, while the serving layer is responsible for answering user queries, similar to the serving layer in the Lambda architecture. Apache Spark is an example of such big data processing platform that combined more than one programming model.

3.3.6 Comparison of Properties of Big Data Models. The comparison between big data models including a batch programming model, stream programming model, NoSQL/SQL models, and message ingestion models is given in Table 1. This comparison is based on five properties, which are data flow (a pattern in data computation implementation), data volume (the size of data), relation (the relationship between the computation implementation of functions), scalability (the capability of increasing resource(s) capacity in response to the future load), and plus and negative points.

As there are different big data models, several big data platforms and services have been developed such as Apache Hadoop, Spark, Storm, Flink, Amazon Kinesis, Azure Stream Analytics, Google Cloud Dataproc, and IBM Streaming Analytics. Rao et al. [105] provided a survey of various big data systems.

3.4 Cloud and Edge Datacenters Level

The cloud and edge infrastructures that provide heterogeneous and distributed compute and storage resources are viable solutions for executing and managing big data workflows, and fulfilling the SLA and QoS requirements defined by users. However, the process of executing and managing such types of workflow in cloud + edge datacenters is a complex resource and data

Table 1. Comparison between Big Data Programming Models

Property	**MapReduce Programming Model**	**Stream Programming Model**		**NoSQL/SQL Model**	**Message Ingestion Model**
		Native	**Micro-batch**		
Data Flow	Static	Streaming		Transactional	Streaming
Data Volume	Known (finite large datasets)	Unknown (infinite continuous events—small data)	Unknown (infinite continuous events—micro-batches (a batch is a finite set of streamed data))	Known (structured data)	Unknown (infinite continuous events —small data)
Relation	Dependent and synchronous computations	Independent asynchronous computations	Bulk synchronous computations	-	Independent and asynchronous computation
Scalability	Horizontal scalability (adding more workers)	Vertical and horizontal scalability (increasing the capacity of workers as well as adding more workers)		NoSQL: Horizontal scalability (using sharding or partitioning technique) SQL: limited scalability (manual)	Horizontal scalability (adding more workers)
Pros (+)	+Extensive and distributed data processing for static data +No need for ingestion system +Estimation of completion time of data processing task	Both: +Extensive and distributed processing for real-time and near real-time data +Store data portion in memory or no store +Low latency (milliseconds for native and seconds for micro-batch model [127]) Native [127]: +No barrier and thus no centralized communication overhead +Low latency during normal execution Micro-batch [127]: +Efficient fault-recovery and scaling due to the use of barriers		SQL: +Multi-row ACID properties +Relational features (e.g. join operations) NoSQL: +Extensive and distributed data processing support with limited flexibility [18] +Support various data types and data speeds +Update schema on the fly	+Extensive and distributed processing for real-time and near real-time data +Different message processing semantics (at-least-once, exactly-once and at-most-once)
Cons (-)	-All data need to be stored in storage system -Redundant and excessive processing -High communication cost -High latency	Both: -Need ingestion system -High overhead Native [127]: -High overheads during adoption Micro-batch [127]: -Need blocking barrier following every batch -Communication overheads		SQL: -No strong support for extensive and distributed data processing [18] -Offline database to update schema NoSQL: -No relational features (e.g., join operations)	-Limit of message size -Overhead -Balancing the data that coming from various data sources

orchestration task. The complexity comes from the composite data flow pattern, various computational models involved in the data pipeline, various big data programming frameworks needed for those computational models, and different types of cloud and edge resources required during the workflow orchestration. The heterogeneous and dynamic nature of cloud + edge resources bring additional challenges (selection of optimal resource types and their configurations, resource failures, and so on), where these challenges also further complicate the workflow-related and data-related challenges, and therefore present a unique cross-layer challenge. The key issue at this level is the real-time selection of the optimal configurations of cloud and edge infrastructures for given heterogeneous workflow components, taking into consideration SLA and QoS requirements defined by workflow users based on the context of the application. This includes the following challenges: cloud platform integration and cloud + edge resources management.

In summary, for managing and executing the big data workflow application, several requirements need to be considered due to complex interaction of the three layers, i.e., (1) big data workflow, (2) the different big data models and different big data analysis applications (such as batch processing, stream processing, SQL, NoSQL, Ingestion), and (3) cloud + edge computing environments. In the next section, we will identify these requirements.

4 REQUIREMENTS OF BIG DATA WORKFLOW IN THE CLOUD

Based on the extensive literature review and study of the characteristics of big data workflow applications, we discuss the key requirements for their orchestration over heterogeneous cloud resources (CPU, Storage, and Software Defined Networking Infrastructure). The heterogeneity at the workflow level (different analytical activities deal with real-time and historical datasets), big data programming model level (batch, stream or hybrid processing), and cloud-level (cloud datacenters and edge resources) leads to diverse requirements that are described as follows:

(1) *Compute/CPU Resources Provisioning Requirement*—To execute tasks/analytic activities related to a big data workflow, a diverse mix and type of compute/CPU resources (e.g., virtual machines, lightweight containers) are required. These resources are provisioned in a static or dynamic way [107] according to the need of such workflow task/activity and type of underlying big data programming model used (e.g., batch processing, stream processing, or hybrid). Provisioning the necessary compute resources for executing big data workflow is not the end of the story. Monitoring and managing those resources in a dynamic execution environment is also needed because those resources are provisioned and released on demand due to changes in data volume and velocity, and resource-level failures [67].
(2) *Storage Requirement*—By taking the decision to move and execute big data workflow using cloud infrastructure, the next decision that will be taken implicitly is moving and storing big data products of such application in the cloud. Thus, we need to intelligently provision the cloud storage to store data and feed the data to different big data programming models at different stages of the workflow execution including, for example, choosing the right cloud storage resource, data location (hence, requires novel indexing and metadata management techniques), and format.
(3) *Data Movement Requirement*—For data residing out of the cloud, such data needs to be transferred to the cloud and stored before being processed by big data workflow. In addition, the stored datasets may reside across different locations and these locations may differ based on geographical deployment of cloud datacenters where compute and storage resources are hosted. So, dynamically transferring these large datasets between compute and storage resources presents new research requirements such as bandwidth allocation, and data transfer latency and throughput management. For example, transferring a large amount of data (i.e., large datasets) needs a high bandwidth. In addition, to an external network (i.e., Internet), dealing with internal networks of the cloud (networks inside the cloud itself) is also needed. The performance of such networks is not the only thing required; dealing with its structure and configuration is also needed. One interesting area of research that will emerge includes how to exploit SDN-based infrastructure within clouds to create more dynamic and flexible data movement techniques and protocols driven by the SLA and QoS needs of workflows.

(4) *Synchronization and Asynchronization Requirement*—In big data workflow, there may exist control and data flow dependencies across analytics tasks. For the dependent tasks, the run-time synchronization is required at both data flow as well as control flow levels. Moreover, the execution of dependent tasks requires dynamic synchronization of the states (e.g., the output of upstream tasks forms the basis of input data to one or more downstream tasks) of upstream and downstream analytic tasks. On the other hand, for independent tasks, no such runtime state (data plus control flow) synchronization requirement exists. In summary, the data and control flow requirement is one of the most important workflow choreography requirements to be considered because it directly impacts the correctness of workflow execution and end-to-end performance, to say the least.
(5) *Analytic Task Scheduling and Execution Requirement*—Provisioning the necessary virtual resources for big data workflow is not the end of the story to running such workflow, scheduling and coordinating the execution of workflow tasks across diverse sets of big data programming models [104] as well as balancing compute resource utilization across the tasks also being required [143]. In addition, partitioning big data workflow into fragments and parallelizing the execution of those fragments using parallelism techniques is important for the scheduling process, which allows it to schedule the partitioned workflow fragments separately on different compute resources to maximize performance and reduce the complexity of scheduling. Moreover, during the execution of a task, the input data for this task is moved to compute resource, the output data is generated, and, in general, data products' provenance is produced [77]. Therefore, tracking and capturing provenance of data is also needed.
(6) *Service Level Agreement Requirement*—The execution of big data workflow may need to meet quality attribute measures defined by users via SLA. These measures, requirements of QoS, are stated in SLA in order to ensure reliable QoS [10]. For example, one quality might be execution deadline, which means the execution of workflow should be completed with strict time constraint (i.e., on or before a deadline). Therefore, we need not only to execute big data workflow in the cloud, but also meet user-defined QoS requirements.
(7) *Security Requirement*—Moving big data computation along with the associated datasets to the cloud imposes the need to secure both data and computation. This introduces a number of challenges that require solutions that go well beyond standard encryption approaches, but include challenges such as private (anonymous) computation, verification of outcomes in a multi-party setting [34], placement of components according to security policies [85], and the like.

    Thus, applying security protection to workflow tasks during their execution and to the data itself provides a high level of security when running such workflow in the cloud.
(8) *Monitoring and Failure-Tolerance Requirement*—Big data workflow comprised of data-intensive tasks and the execution of those tasks is usually a lengthy process. Therefore, monitoring the execution of workflow is needed to ensure that everything is streamlined and executed as anticipated. Moreover, failures could happen at any time during the workflow execution so that handling those failures when they occur or predicting them before they happen is also needed.

5 RESEARCH TAXONOMY FOR ORCHESTRATING BIG DATA WORKFLOW APPLICATIONS

The complexity and dynamic configuration requirements of big data workflow ecosystems call for the need to design and develop new orchestration platforms and techniques aimed at managing: (1) sequence of analytical activities (formed workflow application) that needs to deal with

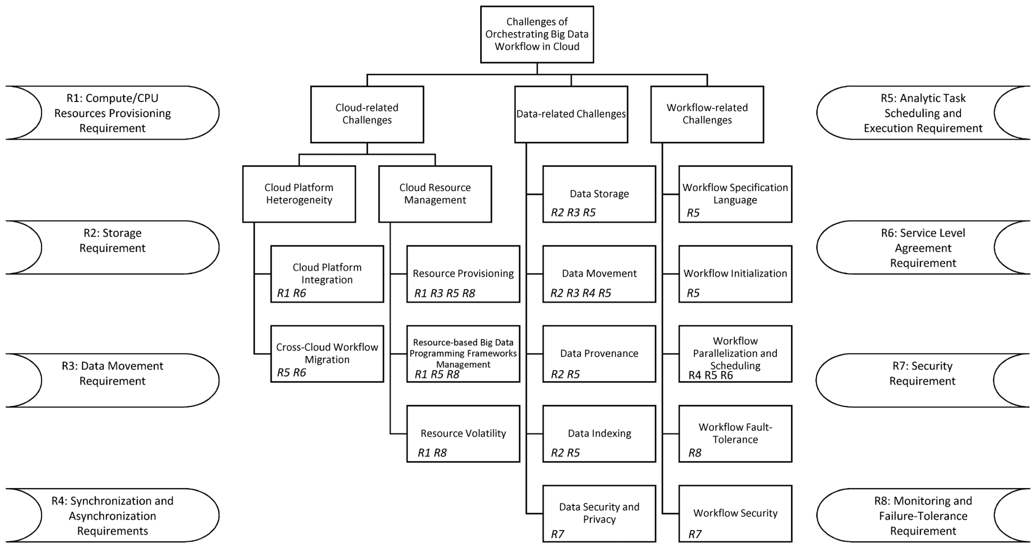
Challenges of Orchestrating Big Data Workflow in Cloud
Cloud-related Challenges
Data-related Challenges
Workflow-related Challenges
R1: Compute/CPU Resources Provisioning Requirement
R2: Storage Requirement
R3: Data Movement Requirement
R4: Synchronization and Asynchronization Requirements
R5: Analytic Task Scheduling and Execution Requirement
R6: Service Level Agreement Requirement
R7: Security Requirement
R8: Monitoring and Failure-Tolerance Requirement
Cloud Platform Heterogeneity
Cloud Resource Management
Cloud Platform Integration R1 R6
Cross-Cloud Workflow Migration R5 R6
Resource Provisioning R1 R3 R5 R8
Resource-based Big Data Programming Frameworks Management R1 R5 R8
Resource Volatility R1 R8
Data Storage R2 R3 R5
Data Movement R2 R3 R4 R5
Data Provenance R2 R5
Data Indexing R2 R5
Data Security and Privacy R7
Workflow Specification Language R5
Workflow Initialization R5
Workflow Parallelization and Scheduling R4 R5 R6
Workflow Fault-Tolerance R8
Workflow Security R7

Fig. 2. A taxonomy of challenges for orchestrating big data workflow in the cloud with the mapping of aforementioned big data workflow requirements to these challenges.

static as well as dynamic datasets generated by various data sources; (2) heterogeneous big data programming models; and (3) heterogeneous cloud resources [104]. The orchestration process contains a set of programming tasks, which are workflow composition, workflow mapping (to map the graph of analytical activities to big data programming platforms and cloud/edge resources), workflow QoS monitoring (to oversee QoS and SLA statistics at runtime for each activity in this graph such as alert delay, load, throughput, utilization), and workflow dynamic reconfiguration (to reconfigure workflows in composite computing infrastructure comprised of cloud, edge, and multiple big data platforms), all for guaranteeing consistency and adaptive management [104]. The requirements posit numerous challenges that do not occur when executing those workflows in conventional computing infrastructure. This section outlines and discusses the research challenges (cloud-related, data-related, and workflow-related challenges) and associated taxonomy with the mapping of big data workflow requirements in the cloud discussed in previous section to these challenges (Figure 2).

5.1 Cloud-related Challenges

The cloud related challenges can be viewed from four dimensions: Cloud Platform Heterogeneity, Resource Management, Data Management and Storage, and Data Security and Privacy.

5.1.1 Cloud Platform Heterogeneity. The cloud platforms offered by different vendors are heterogeneous and vary in their capabilities. Following details challenges associated with this dimension:

(1) *Cloud Platform Integration:* Before provisioning cloud resources, the mapping of big data programming models (that realize different workflow activities) to cloud platforms is required. Each cloud provider defines a specific set of Application Program Interface (API) for supporting such mapping and deployment. This means that the application programming process varies across different cloud providers and for each one of them, the user should learn how to interact with different cloud providers that support heterogeneous APIs [67]. Thus, connecting to multiple cloud platforms is more complex since

the workflow application programmer and/or administrator needs to know the specific programming primitive and patterns relevant to APIs of underlying cloud providers. Accordingly, the user needs to learn several vendor-specific virtualization formats, pricing policies, and other hardware/software configurations, yielding much complex integration challenge. Overall, dealing with integration challenge is complex and requires novel tools, techniques, and API stack for simplifying the mapping and deployment of complex big data workflows to heterogeneous cloud providers.

(2) *Cross-Cloud Workflow Migration:* After mapping and deploying activities of a big data workflow in one cloud, migrating such workflow activities with large datasets to another cloud is a non-trivial process. The users and/or administrators could need to migrate their workflows from one cloud to another because, for example, they might aspire to specific QoS features in the target cloud or better price [67]. In the target cloud, different types of heterogeneous resources (e.g., virtual machines, storage types, network types) are there, and selecting the right number and configurations of resources is a crucial (i.e., remapping and re-deployment) step [67]. Further migrating (mapping + redeploying) workflow activities to other clouds also means moving large datasets and data platforms, which may be a costly and time-consuming task. As a result, the integration challenge in a cloud and/or across multiple clouds (i.e., difficulties of providing a uniform and transparent way to access to different clouds and provision virtual resources from different clouds) is complicated in big data workflows orchestration [67, 104].

5.1.2 Cloud Resource Management. Big data workflow execution in the cloud requires the appropriate selection of cloud resources and their configurations including the provisioning of such virtual resources on demand, and creating and managing those resources as well as coping with the dynamic nature of cloud resources.

Resource Provisioning. As the execution of big data workflow will be carried out in the cloud, the first and important step is selecting the right configuration of virtual resources (virtual machine and/or virtual CPU, storage, and network), which is a challenging decision in case of considering various types of resources offered by various vendors, and becomes even harder when considering different instances from different clouds to achieve the intended goal. Furthermore, when the selection of edge resources comes into the picture, new challenges are being added that include the consideration of diverse edge devices, their hardware features and virtualization support with container technologies, and the conflict SLA and QoS requirements [104]. In addition, the resource selection decision should meet the degree of parallelism needed for data processing tasks composed in a workflow. For example, considering a particular configuration of a cloud such as Google Compute Engine with 18 predefined instance types, it is difficult to find an optimal configuration in order to achieve an optimal execution time, as the configuration selection problem is generally an NP-complete problem [67, 104]. Thereby, with different stages of resource selection, scheduling workflow activities on the selected resources at each stage to run them is also an extremely hard problem (workflow scheduling problem). Also, when considering various resource configurations provided by multiple cloud providers, comparing those configurations to find the best one for a given set of workflow activities is an open research problem as we note in Refs. [103] and [104]; it is not only for workflow activities, but also involving implicitly big data programming frameworks, a cross-layer problem (at IaaS and PaaS-levels) [103]. In other words, the resource configuration search space grows exponentially when we consider each analytical task composing the workflow.

Big data workflows involve various big data workloads, and these workloads have different resource requirements. For batch workloads, the storage requirements dominate; and for streaming workloads, the communication requirements dominate, while for transactional workloads, the computational requirements dominate [103]. Considering different types of workloads in complex workflow scenarios require configuration selection mechanisms to have intelligence in order to assist them in reducing resource contention that can occur due to the interference of workload. This will require determination of those workloads (a.k.a. virtual machines) that can be combined in a same physical environment. Obtaining resource contention information needs both offline benchmarking and real-time SLA monitoring mechanisms.

After the configuration selection, the next step is just to call the cloud provider specific API, which will instantiate the resources we need, for example, virtual machine/CPU instance, storage space, network IPs, and network bandwidth (in case of clouds that support software defined networking). Such a process is not as easy as it seems at first glance because various aspects need to be taken into consideration such as resource location. Big data workflows include multiple data analysis tasks and those tasks are executed in several stages, where each stage might require specific cloud resources. Those resources can be configured differently in order to achieve the intended requirements, but the level of granularity and flexibility is hard to determine [145].

As a result, the problem of resource configuration selection exists across various types of cloud resources since the need here is to allocate resources (virtual machine, storage, network IP, network bandwidth, etc.) to workflow activities and underlying big data programming frameworks. Thus, the allocation of cloud resources at IaaS-level to big data programming frameworks at PaaS-level is no longer a conventional resource maximization or even a time minimization problem; however, it includes simultaneous objectives and configuration dependencies over various IaaS-level resource and big data programming platforms [103].

Resource-based Big Data Programming Frameworks Management. Orchestrating heterogeneous workflow tasks over the cloud requires cloud resources (e.g., virtual CPU, storage, and network) as well as big data programming frameworks (for example Apache Hadoop, Apache Spark, NoSQL). Therefore, the management of PaaS-level big data programming frameworks (that implement various software-based data processing primitives such as batch processing or stream processing) on IaaS-level resources (that provide computing capacity to those frameworks) is needed in the context of big data workflows. Achieving such demand is a complex challenge as it requires determining the optimal approach to automatically select the configurations for both IaaS-level resource and PaaS-level framework to consistently accomplish the anticipated workflow-level SLA requirements, while maximizing the utilization of cloud datacenter resources [103].

Resource Volatility. As mentioned earlier, the loss of the provisioned resources often happens due to different failures [67]. As well, big data workflows consist of complex big data tasks/analytic activities and, thus, the loss of state of analytical processes executed by the big data programming framework could happen at any time. Accordingly, several challenges have emerged as a consequence of the complexity and dynamic nature of cloud resources and big data workflows (i.e., different platform and infrastructure requirements for each workflow task, and dynamic processing requirements of each workflow task, which are determined by either data flow or control flow dependencies).

During the execution of workflow tasks involved in workflow application, we consider the task completed when the following steps are executed successfully before the provisioned virtual resources are being terminated and released: (1) data computation and processing is done and (2) the output data as a result of this computation and processing is stored in temporary or permanent storage. However, the user may, at any time and under any circumstances, terminate the virtual resource while the execution of a workflow task is still under way or the aforementioned steps are not yet completed. This highlights a challenge to deal with failures of virtual resources that originated not from the resources themselves but from user actions. Even after a successful completion of task executions, storing output data products produced as a result of the execution of a big data workflow application is a challenging task, since those data products are big data products and the user in most cases tries to avoid paying for unused virtual resources after completion of execution by terminating and releasing those resources immediately [67]. Moreover, the user might need to add new analytic tools and libraries to virtual machines to be used later on. Those products could be lost in the case of terminating virtual machines (VM) if precautionary actions are not taken. Furthermore, workflow may rely on specific libraries and packages to run, where different tasks might have different dependencies. The volatile nature of cloud resources means that configuring a virtual machine with the required dependencies is actually not a one-time procedure, where such configuration will be lost in the cases of the VM being terminated.

In addition, the problem of resource volatility becomes more complicated in big data workflows when considering the volatility of resources at different levels (VM-level, big data progressing framework-level and workflow task-level). The resource volatility at the VM level is the sense of losing the state of the virtual machine in terms of data stored in Random Access Memory (RAM) and/or non-persistent storage. At the big data programming framework level (such as Apache Hadoop), it is the sense of losing the state of mapper and reducer processes, which we cannot capture at VM level; while at workflow tasks level, it includes the loss of analytic computation completed so far, which may incur additional cost or delay in execution with the best case. Overall, dealing with resource volatility in the context of big data workflows is a more complex task.

5.2 Data-related Challenges

Moving the execution of big data workflow to the cloud means dealing with end-to-end orchestration operations relevant to securing and managing data including storage, movement, provenance, indexing, security, and privacy. Each of these orchestration operations individually is a very challenging task [145] and, to a large extent, this remains a fertile area of research in the context of big data workflows.

5.2.1 Data Storage. As big data workflows support the composition of heterogeneous data processing tasks into data pipelines, the different types of data flow (batches of data or streams of data) associated with different big data programming models that form part of workflows exist. For instance, with message queueing (such as Apache Kafka) and stream processing (e.g., Apache Storm), the streams of data flow into Kafka via Kafka producer or into Storm cluster via spout, while with batch processing (e.g., Apache Hadoop), large datasets should be stored over cloud storage and then fed into Hadoop cluster via HDFS. Since the execution of big data workflows will be carried out in the cloud, the different storage needs for heterogeneous workflow tasks lead to different challenges in dealing with cloud storage resources to satisfy their needs.

With batch processing tasks, these tasks communicate using files (i.e., large files). The one or more output files (output datasets) generated by each analytical task become input datasets to other analytical tasks, and those datasets are passed between tasks using data movement techniques or through shared storage systems [17]. Thus, the large input datasets stored out of cloud must be moved to and stored in the cloud before analytics can start, and the intermediate output datasets generated during the processing as well as the final large output datasets produced upon the completion of processing are required to be put in storage in the cloud, where the data can be thrown out after analytics is done. On the other hand, for stream processing tasks, the analytic result and some input data for provenance can be stored. Accordingly, different storage needs for workflow tasks incur different computing "network and storage" costs, where dealing with that is so complicated as compared with traditional application workflows. Also, choosing which cloud storage resources to use to store data for batch and stream processing tasks has a direct implication on the incurred computing cost. Such selection is a challenging task and becomes more difficult when taking into consideration where the data will be residing, which data format will be used, and where processing will be done.

5.2.2 Data Movement. As the execution of big data workflows will be carried out in the cloud, transferring data to the cloud, within the cloud, and/or between clouds is needed to proceed in execution. The different data flows associated with different big data programming models poses different demands for data movement, so that the problem of data movement is more complicated in the context of big data workflows.

For batch processing tasks, transferring input datasets stored in local machine to cloud storage or data in bulk through hard-disks is required before those tasks are started. Similarly, intermediate datasets produced by those tasks must be moved among execution virtual resources and the outputs resulting from their execution must also be transferred to the following tasks or to cloud storage. Thus, coping with the movement of high volume of historical data to the cloud, and between clouds for batch processing tasks is a non-trivial challenge, because this movement is a costly and time-consuming process as well as having direct implications (in term of expensive execution overhead). Moreover, avoiding both the suspension of some workflow tasks to perform data movements [123] and the waiting time until data is moved to the execution environment are important issues that should be addressed with this challenge.

On the other hand for stream processing, there is no bulk data to be transferred as data is continuously coming from data stream sources and ingesting into data pipelines; however, the streams of data generated by data producers should be moved to the cloud resources where stream processing tasks are executed, and that is incurring data transfer time as well as data transfer cost in case of transferring data between clouds, which are relatively small compared with moving high-volume batch processing of historical data. The challenge here is avoiding or at least minimizing the delay in transferring real-time data as the freshness of this data so important, as well as analytical results for streaming data.

Accordingly, different data flows affect data movement latency (high with batch processing due to moving vast volumes of historical data and low for stream processing as the size of the stream is small), as well as incurring different network costs (high with batch processing and low with stream processing). Hence, the complex problem in data movement is no longer just in moving one type of data flow; but heterogeneous workflow tasks communicate using different types of data flow where each one has its implications on the movement of data in big data workflows. In addition, despite the different types of data flow in data pipeline, transferring the large-scale application data that have been collected and stored across geo-distributed datacenters may be subject to certain constraints (e.g., the data size, the network burden or the General Data Protection Regulation (GDPR))) that determine which of these data can be migrated and which cannot [21, 27, 58]. Thus, the problem of data movement becomes even harder if taking into consideration such data movement constraints and more complex if taking into account several constraints together (such as data size and network burden) when moving data across geo-distributed datacenters during data processing time.

5.2.3 Data Provenance. The data provenance describes the origins and historical derivations of data by means of recording transformation provenance (those transformations that are in charge of creating a certain data element) and data provenance (derivation of a given data element as of which data elements) [49]. An instance of such transformation is big data workflow that generated data product provenance. Dealing with provenance for such workflow is a challenge due to the properties of big data (i.e., 3Vs of big data), the characteristics of the execution environment (highly-distributed and dynamic), the distributed provenance information of data elements, the cost of transferring such information together with data elements (large amount of information), and the complexity of evaluating queries over such information [49].

The track of provenance of historical data deals with a large volume of finite datasets, so that the provenance collection cost/overhead is high and the collecting provenance can be grown larger than the size of data being described. For that, provenance data of historical data is too large, and the storage requirement is becoming an additional challenge. Tracking of provenance of streaming data deals with infinite streams, non-deterministic behavior (e.g., high input rates, delay), stream aggregation (combining multiple streams into one by streaming workloads), ordered sequences (order of streams), and performance requirements (e.g., provenance generation and retrieval) [47], makes it a hard challenge. As well, the fine-grained provenance data generating from small datasets (e.g., streaming data) can be large in size so that the storage requirements [59] and provenance collection overhead are the associated challenge, but the communication overhead is the dominant challenge. For that, this hard challenge becomes even greater since the demand is to tradeoff expressiveness (provenance representation) with a moderate overhead during provenance processing and retrieval.

Moreover, the dynamic and distributed execution environment introduces the demand for capturing and querying distributed provenance of data products, which makes the audit, track, and query of distributed provenance more complex [86]. In addition, distributed transformations of a data item incurs collecting both the provenance of such an item that refers to data and transformations out of each virtual machine that was in use when creating such an item, where the transfer of data items with their provenance means to transfer a large amount of information among virtual resources. That is leading to the need for distributed query solution for big provenance [49]. As a result, the problem of data provenance is complex, but its importance lies in the fact that tracking of data provenance allows understanding and reusing workflows [145].

5.2.4 Data Indexing. It is data structure that aims at creating indexes on datasets in order to accelerate data access as well as data query operations. Data indexing is an issue in big data workflows because (1) each workflow step needs different datasets based on nature of analytic computation; (2) datasets are tremendous, highly dimensional, heterogamous, and complex in stricture [20]; and (3) execution environment (i.e., cloud) is distributed, where all of that complicates the developing of the indexing approach. Moreover, challenges exist in knowing the type of data being indexed and the data structure being used, keeping the cost of creation and the cost of storage space (for storing indexes) low or moderate, and specifying index generation and size.

5.2.5 Data Security and Privacy. In cloud and edge computing, data security remains a major concern. When the execution of big data workflow is performed on cloud/edge resources, big data being processed by workflow tasks will be stored in and accessed from the cloud/edge. Thus, security aspects of workflow data include cryptography (to secure actual data [89]), integrity (to ensure data consistency and accuracy), access control (to enforce restrictions on access to data), authentication (to verify the identity of an individual with or without disclosing such identity [53, 54]), data masking (to replace sensitive data with masked data), tokenization (to replace original data with random value of the same data format [89]), and privacy (to restrict the collection, sharing, and use of data). Since investigation of these aspects is a large topic in itself and is beyond the scope of this article, we list some of security issues and briefly review them along with the related approaches in Appendix D.

5.3 Workflow-related Challenges

5.3.1 Workflow Specification Language. Workflow specification language defines workflow structure and its task. For big data, there are different big data models available such as batch processing (MapReduce), stream processing, SQL, NoSQL, where each one of them have their own way to specify computation, so that further functionality and flexibility in the specification of workflow is required to support those models. Consequently, the hard challenge here is to create a workflow-level specification language that can be more human-friendly and can be automatically transformed to programming model specific specification language (e.g., MapReduce in the context of Apache Hadoop, continuous query operators in the context of Apache Storm, relational queries in the context of SQL, non-relational queries in NoSQL databases). Moreover, this challenge becomes more complicated if the specifications of cloud resource and big data management are taken into consideration. The former challenge is in specifying the cloud resource specification as part of the workflow specification language at least at high level and that could be in terms of QoS, performance, security/privacy constraints. The latter challenge is in specifying big data management specification, also as part of specification language, at least at high level, and that could be in terms of data format, storage constraints, or data movement restrictions. In addition, for Multicloud architecture, a standard specification language for big data workflows is needed to make such workflows portable and scalable across multiple clouds.

5.3.2 Workflow Initialization. Workflow initialization aims to divide a workflow into several small parts called workflow fragments (or fragments for short) to allow parallel and distributed processing, where each fragment contains part of the workflow activities and their data dependencies [76, 77]. It could be a constraint-based process to take into account some constraints such as compute resources or minimizing data transfer during the partitioning of a workflow. Since big data workflows include multiple data analysis tasks and those tasks are executed over virtual resources provisioned from one or more clouds in a parallel and distributed manner, such an initialization process is needed. In other words, workflow initialization is needed for executing big data workflows in the cloud in a parallel and distributed manner.

Workflow initialization is a non-trivial task since it necessitates to take into account the task and data dependencies within the workflow and to avoid cross dependency. It becomes harder if considering the aspects of data management (storage and indexing). For the data storage aspect, we need to consider the different needs of storage resources for heterogeneous workflow tasks during the partitioning process, so that fragments of workflow respect these needs. For the data indexing aspect, we need to consider the index data for datasets during the partitioning process since each workflow step requires various datasets based on the nature of analytic computation, so that the data required for these workflow steps can be searched and retrieved quickly.

Furthermore, this challenge becomes more complicated if other restrictions are taken into account, for example, multisite execution, data transfer, storage resource constraints, or balancing the activities of workflow in each workflow fragment whilst lessening the linked edges amongst various workflow fragments [77].

5.3.3 Workflow Parallelization and Scheduling. After initializing the workflow, the partitioned workflow fragments are parallelized and scheduled on cloud resources in order to be executed on those resources. For workflow parallelization, various techniques are utilized in order to produce concrete executable workflow tasks for the execution plan [77]. The workflow parallelization results included in the workflow execution plan is a decision of parallelizing workflow tasks to execute them in parallel. For big data workflows, they are parallelizable across all two levels: big data programming framework level and workflow activity level. At the big data programming framework level, the frameworks (e.g., Apache Hadoop, Apache Storm) are workflows themselves, for example, workflow of Map/Reduce tasks in Apache Hadoop or workflow of spout and bolts tasks in Apache Storm. While at workflow activity level, each activity is heterogeneous and mapped to a different machine.

The workflow scheduling needs to cater for above super-workflows and then find an optimal resource allocation plan for lower level cloud resources, which is an extremely hard research problem. This complexity comes from several aspects that have to be taken into consideration to create an efficient scheduling plan. This plan is aimed at balancing resource utilization across sub-workflows involved in big data workflow and making the execution complete to achieve the desired objectives. It will also be revised in response to unexpected changes occurring at runtime such as changes in data velocity and resource availability. The heterogeneity of data analysis tasks involved in big data workflows complicates the situation. Considering the location of data during the task scheduling period is important [77]. In other words, task scheduling has to be aware of the location of data to minimize data movement. Moreover, the user quality of service requirements needs to be considered. Furthermore, the use of Multicloud architecture is a complex aspect since it necessitates to be aware of the arrangement of resources and big data programming frameworks in this architecture in order to map the fragmented workflow parts or tasks to available workers, in addition to utilize resources in this architecture by considering data location during task scheduling.

Furthermore, when workflow scheduling utilizes edge resources, new challenges come into the picture to efficiently map and deploy data analysis tasks on resource constrained edge devices including the achievement of three core properties of a container, which are isolation, orchestration, and scheduling [105]. The lightweight hardware with lightweight containerization software is needed [130]. Finding the optimal resource selection and allocation plan for executing data analysis tasks at edge resources should take into account the characteristics and hardware constraints of edge devices, and the heterogeneity of these tasks, where this plan could be part of a full allocation plan for cloud and edge resources. The other challenge is adapting heterogeneous big data workloads (i.e., data analysis tasks) from VM-based workload to container-based workloads for container platform, Kubernetes. This also includes the challenge of creating efficient container images for those workloads. Moreover, managing and monitoring the containers for big data workloads over edge resources with Kubernetes is complicated due to the dynamic nature of edge resources and their changing performance, and the need to define runtime configuration and deploy them into the container environment. This container management process becomes even harder with the need to maintain SLA and QoS requirements on those constrained resources (such as execution cots and data processing throughputs), and to respond to unexpected changes at runtime [104]. Overall, the complexity of orchestration is increased when scheduling is considered in an edge environment virtualized using lightweight containers.

Finding optimal virtual resources to allocate them to workflow tasks and underlying big data programming frameworks, and the optimal configurations for both IaaS-level resource and PaaS-level big data programming frameworks helps to achieve SLA scheduling.

5.3.4 Workflow Fault-Tolerance. Workflow fault-tolerance intends to handle failures that occur during the execution of workflow and assure its availability and reliability [77]. Since big data workflow is complex, dynamic, and heterogeneous (the complexity and dynamism of big data ecosystems), and its execution is usually a lengthy process and will be carried-out in a dynamic environment, the failures may happen at any time for numerous reasons such as the change in execution environment, the loss of compute resource, error during analytical task exaction, or unhandled errors during task execution. Thus, failure management in workflows is much more complicated as things can go wrong at any level (workflow-level, big data processing-level, and cloud-level). It becomes harder and harder with big data workflow consisting of data- and compute-intensive tasks along with the need to predict failures and accordingly take appropriate actions in order to avoid additional expensive costs that could be incurred if failures occur such as re-computation and re-transferring data costs. As a result, developing a holistic failure-management/tolerance technique is a challenging process, and most likely ends with a suite of failure management algorithms specific to each workflow activity type.

5.3.5 Workflow Security. In the context of big data workflow, securing big data [30] is not the whole story, it is considered as a part in preserving workflow security. The other part is guaranteeing the security of workflow logic and computation. As big data workflow is data-centric, ensuring the security and integrity of processing or computation carried out on big data in addition to data security are the main challenges. The lack of interoperability is a particular issue since the underlying execution environment used for running such workflow is distributed and heterogeneous by nature [67]. Moreover, with the difficulties of managing authentication and authorization in such workflows, preserving such levels of security becomes even harder and more challenging. The heterogeneous data processing involved in workflow may require different security needs adding more complexity and makes ensuring security at workflow level a complex task [85].

6 CURRENT APPROACHES AND TECHNIQUES

Presenting several orchestrating big data workflow in the cloud is important, but knowing how to resolve them is crucial. This section reviews the current approaches and techniques related to the presented research taxonomy.

6.1 Cloud Platform Integration

The aim of cloud platform integration is to mask the heterogeneity among different cloud platforms offered by various cloud providers in order to provide a uniform way to access clouds of various vendors as well as to provision, consume, and manage resources from different clouds. There are two following generic approaches to resolve the incompatibilities between different cloud platforms [51].

Standardization Approach. This approach intends to standardize interfaces in each service level of cloud computing so that cloud applications and resources are provisioned, deployed, and managed independently of specific platform environments. It is an efficient approach to accomplish cloud integration (cloud interoperability and portability), but it is very complicated for different cloud platforms to agree on common standards.

Intermediation Approach. This approach intends to provide an intermediate layer (middleware service or platform) that hides the proprietary APIs of cloud providers. It achieves that by dealing with several vendor-specific connection protocols and APIs, and vendor-specific provisioning interfaces as well as all stages of the software development lifecycle. As the integration challenge is raised, some recent efforts such as SimpleCloud and mOSAIC [88] have attempted to mask the API heterogeneity between different cloud platforms by providing uniform, multi-provider compatible APIs. Libraries such as jClouds enable access, provisioning, and management of resources from different clouds, and mOSAIC offers the developers an abstraction from native APIs by implementing several API layers [51]. Therefore, this approach provides a quick and easy way to have access to different clouds supported by the selected toolkit.

6.2 Cross-Cloud Workflow Migration

The migration process here aims to migrate the workflow completely or part of it (in terms sub-workflows or analytic activities) from one cloud system to another, targeting several optimization requirements such as improving performance, reducing execution cost and time, and achieving specific QoS features. Figure 3 shows the classification of cross-cloud migration approaches for

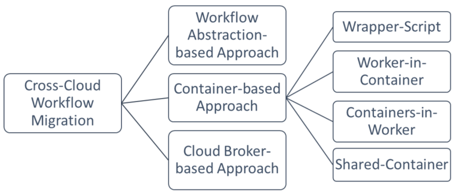

Fig. 3. Classification of cross-cloud migration approaches for workflow.

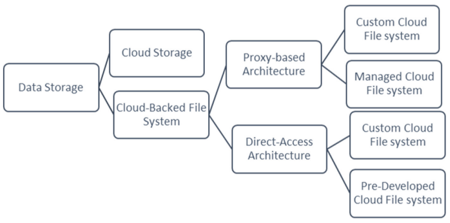

Fig. 4. Classification of data storage approaches for workflow.

workflow. As seen from this figure, the three approaches for migrating workloads (i.e., workflow and its analytic activities) between different execution environments are workflow abstraction-based, cloud broker-based, and container-based approaches.

Workflow Abstraction-based Approach. This approach aims to describe abstract data-intensive workflows, enabling the portability of these workflows across diverse execution platforms. The abstract model is used to define data-intensive workflow and remove the details of target execution platforms and the steps of data handling [38, 40]. Makeflow [6], Asterism DIaaS [38], and dispel4py [39, 40] are examples of workflow abstraction models that are not mainly designed to support the heterogeneity and dynamism of big data workflows, however, they can be elaborated in abstraction of those workflows.

Cloud Broker-based Approach. This approach provides the ability to run workflow applications in intercloud environments. It acts as mediator among users of workflow and providers of cloud systems, helping in the selection of target cloud(s), accessing this/those cloud(s), and achieving user-defined SLA and QoS requirements [65, 66].

Container-based Approach. This approach exploits containerization technology (e.g., Docker, udocker [50], Singularity [72]) to provide the ability to quickly and efficiently build and deploy workflows (sub-workflows or workflow activities) across cloud computing systems by encapsulating compute resources and delivering a user-defined execution environment [45, 101, 149]. A container packs only the libraries and packages (i.e., full software stack) needed by sub-workflow or workflow activity [101]. By doing that, the workflow portability and migration are improved, allowing seamless and agentless migration of workflows across diverse cloud infrastructures.

6.3 Resource Provisioning

Resource provisioning aims to select and provision the cloud resources that will be used to execute the tasks (i.e., big data workflow fragments or tasks). There are two following approaches to resource provisioning.

Static Resource Provisioning Approach. This approach takes the decision of provisioning virtual resources that are required to run workflow fragments or tasks before the execution of workflow. It is not able to dynamically scale resource in or out [107]. The provisioned resources are fixed, and they are the only resources available during the whole period of workflow execution. Thus, such an approach is suitable to be used in cases where the demand of the workflow is predicted and fixed in term of resources.

Dynamic Resource Provisioning Approach. In contrast, this approach takes the decision of provisioning resources during the execution of workflow or at runtime. It decides which resource types and configurations are most suitable, and when to add or remove resources according to the demands of the workflow. In other words, this approach is taking all decisions or refining initial ones at runtime and determining which virtual resources need to keep running and active, which resources should be provisioned, and which resources from the provisioned resources should be deprovisioned as the workflow execution progresses. This approach aims to avoid under-provisioning because of its implication on performance (lowers performance) and over-provisioning because of its implication on cost and system utilization (increase the cost and lowers system utilization).

6.4 Resource Volatility

In any environment, there is a possibility of losing these resources or the state of analytical processes executed by big data programming framework at any time due to different failures. Mitigating such failures need to be carried out at different levels (VM level, big data progressing framework level, and workflow tasks level). For each level, a corresponding approach is needed to mitigate those failures that occur at this level, achieving that without ignoring the consideration of resource consumption and performance efficiency. Therefore, there are three level-based approaches: at VM-level, at data processing level, and at workflow-level.

—VM-level Mitigation Approach: This approach aims to mitigate the failure and the loss of the state of virtual machines in terms of data stored in RAM and/or non-persistent storage. Examples of techniques under this approach are replication approaches based on active/active or active/passive mode. Checkpointing is a common technique that can be used to save or replicate the state of VM (periodically or on-demand), and then mitigate failures by recovering from a stored or replicated state [33, 115]. VM workload consolidation-based fault-tolerance technique is another technique used to improve the VM's reliability [74].

—Big Data Processing Framework-level Mitigation Approach: This approach aims to mitigate the failure and the loss of the state of computational units/processes within the big data processing system (such as with Apache Hadoop—the sense of losing the state of mapper and reducer processes, which we cannot capture at VM level). Examples of techniques used to recover from the failures of data processing tasks are byzantine fault tolerance (in MapReduce [28]), replication-based fault tolerance (in MapReduce [80]), and rollback recovery (in dataflow systems [62]).

—Workflow Task-level Mitigation Approach: This approach aims to mitigate the failure and the loss of workflow tasks including the loss of analytic computation completed so far, which may incur additional cost or delay in execution with the best case. Workflow task-level techniques (either reactive or proactive) can be to handle task failures occurring during the execution of workflow.

6.5 Data Storage

Big data workflow is comprised of a set of data-intensive tasks, which communicate using large files (large datasets). These large files should be stored in the cloud since the execution of big data workflows will be carried out in the cloud, and be passed among workflow tasks using data movement techniques or shared storage systems [17]. Figure 4 shows the classification of data storage approaches for workflow.

Cloud Storage. This is a storage service offered by cloud providers. This approach requires the workflow management system to manage data on the cloud storage service [17]. However, the reliability of data stored in a cloud storage system could be an issue with this approach [93].

Shared Cloud-backed File System. It intends to deploy shared file systems in the cloud [17], where the backend can be single cloud (by utilizing single cloud storage service) or cloud-of-clouds (by utilizing multiple cloud storage services) . It resolves the storing data problem in a generic way and follows either a proxy-based architectural model or direct-access architectural model [12]. The descriptions of these models are as follows:

- Proxy-based Model: In this model, the proxy is implementing the core functionality of the file system and interacting with cloud storage in order to store and retrieve files [12]. With the single file system, the single point of failure and performance bottleneck are issues with this model [12], while the parallel file system addresses those issues. The file system following this model can be:
  - Custom Cloud File System: The aim here is to build a custom shared file system for workflow tasks. For example, a file system can be hosted in extra VM provisioned from a particular cloud platform/infrastructure, and the other provisioned VMs, i.e., worker nodes can mount such a file system as a local drive/volume [17]. In addition, parallel file systems can be hosted by several VMs in case better performance is required [17].
  - Managed Cloud File System: The aim here is to select and use one of the shared file system options offered by cloud providers.
- Direct-access Model: In this model, there is no proxy and the access to the cloud storage is direct. Also, with this model, the single point of failure is no longer an issue, but it becomes hard to offer file sharing in a controlled manner since the convenient rendezvous point for synchronization is missed [12]. The file system following this model can be:
  - Custom Cloud File System: The aim here is to build a custom shared file system for workflow tasks without the interposition of a proxy.
  - Pre-developed Cloud File System: The aim here is to select and use an existing shared file system.

6.6 Data Movement

By moving the execution of big data workflow to the cloud, the working datasets should be moved to the cloud as well. These datasets are large datasets and moving or transferring them is an expensive task. In the literature, several research works have proposed various approaches to tackle the problem of data movement for data-intensive workflows. The classification of data movement for workflow is depicted in Figure 5.

Data Transfer. This approach intends to transfer data with minimal data transfer time. The heterogeneity and instability of the cloud network affect this transfer [123]. The following are three different techniques to achieve the lowest data transfer time [95]:

- Data Parallelism: The ability of a service to process data chunks in parallel with minimal performance loss. Such ability includes the processing of independent data on various compute resources.
- Data Streaming: This technique intends to enable data transport among workflow fragment/tasks through the use of data streams. That allows support for high-throughput and low latency.
- Data Throttling: This technique intends to determine and control the arrival time and the rate of data transfer as opposed to the movement of data from one place to another as quickly as possible. As an alternative of transferring data to a task, this technique can be used to delay data transfer or transfer data using lower capacity links in order to allocate resources to serve other crucial tasks.

Data Locality. Since the working datasets for big data workflow are huge, moving those datasets among compute resources provisioned from multiple clouds is costly and time-consuming. This approach aimed at minimizing data movement by means of moving the computation in proximity of data. The different techniques to exploit such an approach are as follows:

— Task Centric: This technique aims to move workflow tasks toward data without considering the interest of workers. The locality of data is exploited by schedulers to map tasks to compute resources in which tasks are being executed on a compute resource that is in or close to the location of data. Task clustering is a method that aims to group small workflow tasks together as one executable unit for eliminating data movement overhead (and, by the way, removing the overhead of executing those small tasks). By doing grouping of tasks, the intermediate results generated by each grouped task remains in the same virtual resource (i.e., VM), which allows other grouped tasks to locally access the result. A special case of task clustering is spatial clustering. With this method, a workflow task is created by relying on the spatial relationship of files in datasets [95]. It groups workflow tasks into clusters based on spatial proximity, where each cluster contains a subset of tasks and is assigned to one execution site. Network-aware task distribution is a method exploited by a scheduler to mix data localization and geo-distributed datacenter data transfer (network bandwidth) requirements to tackle the data movement problem for a large-scale application whose data has been collected and stored across geo-distributed datacenters and is subject to certain constraints (e.g., the GDPR) [21, 27, 57, 58, 63].

— Worker Centric: This technique aims not only to exploit the locality of data, but also to take into consideration the interests of workers on executing computation. The idle worker takes the intuitive and expresses its interest to execute a workflow task; in that case, the scheduler chooses the best task for this worker by exploiting locality in data accesses.

Co-location of Data and Computation. Instead of moving data to compute nodes or bringing computation to the data, co-locating data and computation is a viable solution; it addresses the data problem as part of the resource utilization strategy. Thus, this approach aims to combine compute and data management for tackling the problem of data movement to minimize the overhead and scalability for forthcoming exascale environments to achieve better resources utilization.

6.7 Data Provenance

As mentioned earlier, the provenance data for big data workflow represents the execution behavior of such workflows, which allows tracing the data-flow generation [29]. To provenance data, there are two following approaches based on granularity level [49].

Coarse-grained Provenance. It is a control-flow-based approach that does not peer into data-flow inside transformations and handles them as black boxes, so that for a given transformation, it records the elements of data that are inputs and outputs of such a transformation. For instance, with word count transformation and by considering documents as single data units, this approach deliberates all documents as a pair (w, c) provenance. The graph structure is usually used to represent such information in which data elements are linked to provenance transformations that generated or consumed those elements.

Fine-grained Provenance. It is a data-flow-based approach that peers into data-flow inside transformations to provide insight information. In other words, this approach exposes the transformation processing logic as a result of modeling the significant parts of inputs in the derivation of a specific output data element. For instance, with word count transformation and by considering

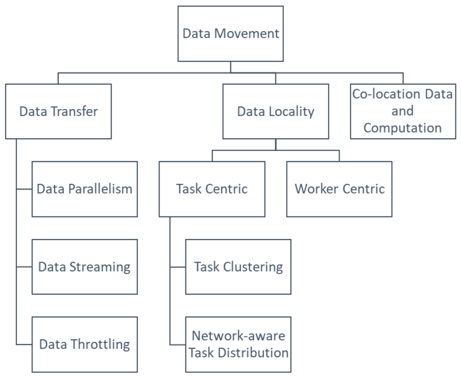

Fig. 5. Classification of data movement approaches for workflow.

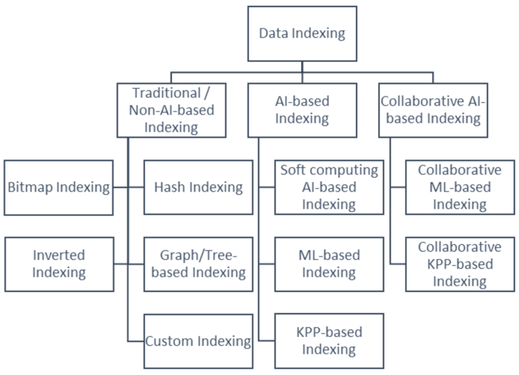
Data Indexing
Traditional / Non-AI-based Indexing
AI-based Indexing
Collaborative AI-based Indexing
Bitmap Indexing
Hash Indexing
Inverted Indexing
Graph/Tree-based Indexing
Custom Indexing
Soft computing AI-based Indexing
ML-based Indexing
KPP-based Indexing
Collaborative ML-based Indexing
Collaborative KPP-based Indexing

Fig. 6. Classification of data indexing approaches for workflow.

documents as single data units, this approach deliberates input documents that contain the word w as provenance of a pair (w, c).

6.8 Data Indexing

The aim of data indexing is to accelerate data access as well as data query operations; but, it comes at an extra cost for both data index creation operations and data writing operations, and additional storage space required for storing these indexes. Various indexing approaches have been reviewed and investigated for big data requirements in the literature [3, 18, 43]. The classification of data indexing for big data in big data workflow is depicted in Figure 6.

Traditional / Non-AI-based Indexing. With this approach, neither the meaning of the data element nor the relationship among words is included in the index formation. That means the formation of indexes is dependant on the cover-known patterns (i.e., most searched and retrieved data elements) in a given dataset. Hash indexing is an efficient strategy for data access and retrieval in a high-dimensional data context as it is able to detect duplication of data in a big dataset. Bitmap Indexing is a strategy that uses bitmap data structure for storing data and then retrieving it quickly. It works nicely with low-cardinality columns as well as being considerably appropriate for big data analysis along with low data storage space. Graph/Tree-based indexing strategy is a strategy that uses the index of more complex data structures to make data indexes to enhance the performance since the bitmap data structure is feeble in transaction processing. Examples of such data structures used for indexing data are B-Tree, B+-Tree, R-Tree, Log-Structured Merge (LSM)-Tree, and bLSM (LSM with B-Trees and log structured methods). In case of storing the big data of workflow application by using SQL model with many relational database systems and/or NoSQL model with Cassandra, BigTable, or/and HBase, this approach is followed but with different data strictures. Indeed, many relational database systems used B-Tree data structure while aforementioned NoSQL database management systems are using LSM-Tree data structure [121] to support data indexing. Inverted Indexing strategy intends to enhance the full-text searching capability by the use of an inverted index data structure to store the mapping of content (e.g., numbers, word sequences) to its location in document database. Custom indexing strategy intends to create multiple field indexing by replying on either random or user-defined indexes. Generalized Inverted Index (GIN) and Generalized Search Tree (GiST) are two types of custom indexing [3].

AI-based Indexing. This approach is able to discover unknown big data behavior by utilizing a knowledge base, providing efficient data indexing, and thus effective data search and retrieval.

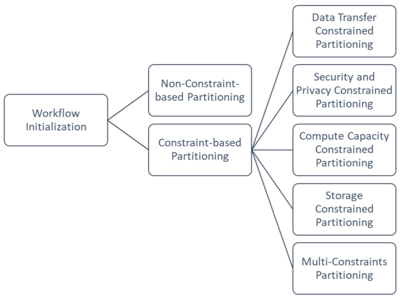
Workflow Initialization
Non-Constraint-based Partitioning
Constraint-based Partitioning
Data Transfer Constrained Partitioning
Security and Privacy Constrained Partitioning
Compute Capacity Constrained Partitioning
Storage Constrained Partitioning
Multi-Constraints Partitioning

Fig. 7. Classification of workflow initialization approaches.

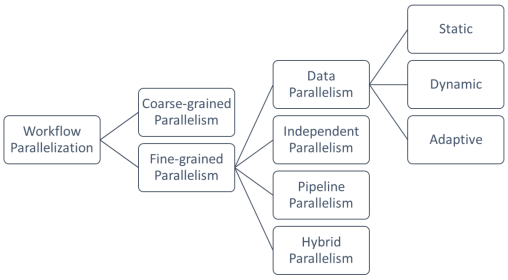

Fig. 8. Classification of workflow parallelization techniques.

However, it needs more time compared with a non-AI indexing approach to answer the search query in general. Soft computing AI-based indexing techniques blend fuzzy set and neural computing methods for indexing data, while Machine Learning (ML)-based indexing techniques improve data indexing by utilizing machine learning methods such as manifold learning. Knowledge Representation and Reasoning (KRR)-based indexing achieves that using semantic ontology.

Collaborative AI-based Indexing. This approach enhances the accuracy of data indexing and the efficiency of search by relying on collaborative artificial intelligence, aimed at providing greater cooperative data indexing solutions. With this approach, collaborative ML-based indexing and collaborative KRR-based indexing methods are provided that relate individual and cooperative decision-making to index big data.

6.9 Workflow Specification Language

Workflow specification language is used to describe the structure of workflow and its tasks to allow interpreting and executing the specification. There are two approaches that we can consider here for specification language: generic and custom. In the generic approach, the generic (general-purpose) workflow specification language is properly selected and used to describe the big data workflow. In the custom approach, the new workflow specification language is designed to describe big data workflows. Such a specification language limits the portability and scalability capabilities for workflow across a variety of execution environments.

6.10 Workflow Initialization

Workflow initialization aims to partition (with or without constraint) a workflow into fragments to parallelize the execution of those fragments over provisioned compute resources. Since big data workflow is composed of data-intensive tasks, parallelizing the execution of these tasks needs partitioning of such workflow into fragments. The approaches of workflow initialization can be classified as either non-constraint-based partitioning or constraint-based partitioning. This classification is depicted in Figure 7.

Non-Constraint-based Partitioning. This approach decomposes a workflow into smaller fragments to allow distribution of those fragments among compute resources for parallel execution. It considers the task and data dependencies within the workflow and avoids cross dependency; no other constraints are taken into account. Thus, the decision of partition is made based on task and data dependencies and not based on the capacity of compute resources or the cost of data movement.

Constraint-based Partitioning. This approach partitions a workflow into smaller fragments, taking into consideration the defined constraint to allow distribution of those fragments among compute resources for parallel execution. It not only considers the task and data dependencies within the workflow, and avoid cross dependency, but also any other constraint that is defined. There are five following techniques to support constraint-based partitioning.

(1) Data Transfer Constrained Partitioning—This technique aims to minimize the amount of data to be moved among fragments of a workflow [76]. By considering the cost of transferring data between fragments that will be executed in one site or a multisite as a partitioning constraint, the workflow will be decomposed in such a way that minimizes data transfer so as to reduce the total execution time.
(2) Security and Privacy Constrained Partitioning—This technique aims to partition a workflow into fragments under security and privacy restrictions. For instance, a workflow may contain a critical activity that requires execution to be done at a trusted cloud site, so this workflow will be partitioned in such a way that this activity and its following activities for processing output data must be designated to the same fragment, and the others will be designated to another fragment.
(3) Compute Capacity Constrained Partitioning—This technique partitions a workflow into fragments according to compute resource configurations. The different configurations of compute resource in one cloud or heterogeneous multisite cloud configurations can be used to adapt workflow partitioning [76]. For example, some tasks of a workflow may need more computing capacity than other tasks, so that those tasks will be assigned to available compute-intensive resources or to the cloud site that has more compute capacity.
(4) Storage Constrained Partitioning—This technique aims to respect storage constraints during partition of a workflow into fragments [24].
(5) Multi-Constraints Partitioning—This technique aims to respect multiple factors or constraints in the process of partitioning a workflow.

6.11 Workflow Parallelization and Scheduling

Following the classification of workflow parallelization techniques presented by Ref. [77], the two parallelization techniques based on the level of parallelism are coarse-grained parallelism and fine-grained parallelism [77]. This classification is depicted in Figure 8.

Coarse-grained Parallelism. This approach achieves parallelization at the level of workflow. It is crucial to meta-workflow execution or parameter sweep workflow execution. For meta-workflow, this technique parallelizes the execution of independent sub-workflows composed of such workflow by submitting them to corresponding workflow engines. In a parameter sweep workflow execution, each set of input parameter values results in an independent sub-workflow.

Fine-grained Parallelism. This approach achieves parallelization at activity level within a workflow or a sub-workflow, where different activities will be executed in parallel. At this level, there are different types of parallelism to handle within an activity and between activities. For parallelism within an activity, data parallelism is used; for parallelism between activates, independent parallelism and pipeline parallelism are used; and for higher degrees of parallelism, hybrid parallelism is used. Following are their descriptions [77]:

—Data Parallelization: This type handles parallelism within an activity. To achieve such parallelism, it needs to have various tasks perform the same activity and each one of them processes different chunks of input data in a various compute node. Thus, the resultant data is partitioned since the input data is already partitioned. This partitioned result (output

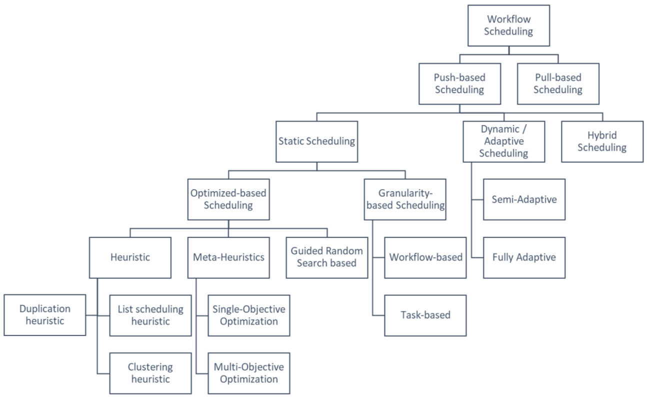

Fig. 9. Classification of workflow scheduling techniques.

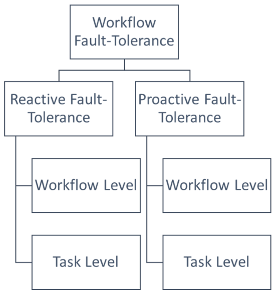

Fig. 10. Classification of workflow fault-tolerance techniques.

data) could be input data for data parallelism for the next activities or be combined in order to produce a single result. This type of parallelism can be static, where the number of data portions is fixed and specified prior to the execution; dynamic, where the number of data portions is identified at runtime; and adaptive, where the number of data portions is automatically modified to the execution environment.

—Independent Parallelism: This type handles parallelism between independent activities of a workflow. To achieve such parallelism, workflow should have at least two or more independent fragments of activities and the activities of each fragment have no data dependencies with activities of other fragments. In addition, those independent activities need to be identified in order to be executed in parallel.

—Pipeline Parallelization: This type handles parallelism between dependent activities. These activities with a popular type of relationship among activities (i.e. producer-consumer relationship) can be parallel executed in pipeline fashion, where the output of one data portion of one activity is the input of the following dependent activities. By exploiting this type of parallelism, the consumption of data portions is performed as soon as those portions are ready.

—Hybrid Parallelism: This type combines three types of parallelism in order to achieve higher degrees of parallelism. It applies data parallelism within each activity, then independent parallelism between independent activities, and lastly pipeline parallelism between dependent activities.

After parallelizing the activities of big data workflow, these activities should be scheduled on cloud resources for execution. Figure 9 shows the classification of workflow scheduling techniques. The techniques of workflow scheduling can be categorized into push-based and pull-based scheduling.

Push-based Scheduling. This technique allows scheduling tasks of workflow among compute resources by pushing them to available resources. The scheduler maps workflow tasks to resources according to the generated scheduling plan. By following task-centric, the scheduling techniques are as follows:

—Static scheduling: This technique generates and assembles schedules that allocate all tasks of workflow to compute nodes prior to the execution of workflow and these schedules (i.e., scheduling plan) are strictly observed during the whole execution [16, 77]. Since the scheduling decision is made before execution of workflow, this technique produces little overhead at runtime. It is efficient and achieves good results when the execution environment experiences little change, i.e., controllable or homogeneous compute environments. With execution environments that vary greatly, it is very hard to achieve load-balance, and with variations in resource performance, the overall execution time will be strongly impaired [16, 77]. There are various scheduling techniques that can be classified into the following categories:

- Granularity-based scheduling: In this category, the process of scheduling workflow is based on granularity level. The techniques/methods are [77]:
  - Workflow-based: This technique maps the partitioned fragments of workflow to compute resources. It is a preferable technique used for data-intensive applications because the overhead of transferring data between fragments is less than transferring data between tasks.
  - Task-based: This technique maps the tasks of workflow directly to compute resources.
- Optimized-based scheduling: In this category, the scheduling problem is considered as an optimization problem. The techniques/methods are [77, 137]:
  - Heuristic: There are three types of heuristics that have been proposed. List scheduling heuristic constructs a scheduling list for tasks that will be scheduled by appointing those tasks some priorities, and then sorting them in accordance with the assigned priorities, then performing "task selection" and "resource selection" steps recurrently until all tasks in the directed acyclic graph are scheduled, where in the "task selection" step, the head of the scheduling list is selected (i.e., first task) and in the "resource selection" step, the task is allocated to the selected resource. Clustering heuristic focuses on optimizing the time of transmission among data dependent tasks. A general clustering heuristic is comprised of two phases, which are clustering (to map tasks to clusters) and ordering (to order tasks that belong to the same cluster). Similarly, duplication heuristics focus on optimising the transmission time using duplication of tasks.
  - Meta-Heuristics: The scheduling algorithm uses a global search oriented meta-heuristic to find a very good solution quickly and efficiently. The most used meta-heuristic for task scheduling problems is the genetic algorithm. Other meta-heuristics are investigated and used such as ant colony optimization and particle swarm optimization. There are two types of meta-heuristics algorithms: (1) single-objective optimization for optimizing the value of single objective function either by minimizing or maximizing it, and (2) multi-objective optimization for more complex problems with more than one conflicting objective that it is obliged to optimize simultaneously, where it provides alternative optimal solutions by trading off between those objectives [120].
  - Guided random search based: This technique schedules tasks randomly.

— Adaptive/Dynamic scheduling: This technique generates a scheduling plan that maps workflow tasks to compute resources at runtime by monitoring execution infrastructure [16, 77]. Such a plan is adjusted continually during the execution of workflow according to the perceived changes. It is a suitable technique for use when workflow tasks exist in a highly dynamic environment or when the amount of work for those tasks is hard to estimate. This technique can be either:

- Semi-Adaptive: In this type, the scheduler maps tasks to compute resources during workflow execution in accordance to the online performance statistics of the resource.
- Full-Adaptive: In this type, the scheduler schedules tasks onto compute resources during workflow execution in accordance to the online performance statistics of the resource,

plus specific task requirements and characteristics. For example, a full-adaptive scheduler may map a workflow task with a high degree of parallelism on a compute resource that has multiple threads.

—Hybrid scheduling: This approach combines static and dynamic scheduling to gain the advantages of both in order to provide better performance than just using one or the other [77]. For instance, the static scheduling can be used to schedule part of workflow tasks (e.g., there is enough information for them) and the remaining tasks can be scheduled at runtime using dynamic scheduling [77].

Pull-based Scheduling. This technique intends to exploit the interest of worker/node in scheduling a task when it is idle by allowing the worker to request from a scheduler to schedule a task and the scheduler pulls the best task for this worker from among available tasks and maps such tasks to it.

6.12 Workflow Fault-tolerance

Big data workflow composed multiple data-intensive tasks and the execution of those tasks is usually a lengthy process, so that failures or errors could happen at any time during the execution period. Workflow fault-tolerance intends to handle failures occurring during the execution of workflow and assure its availability and reliability [77]. Figure 10 shows the classification of workflow fault-tolerance techniques.

Reactive Fault-Tolerance. This technique aimed at minimizing the impact of failures after their occurrence. To achieve that, there are numerous fault-tolerance techniques such as checkpoint/restart, replay (replication), and retry (task resubmission). This technique can resolve the faults at either:

—Workflow level: As the name suggests, detecting failures is carried out at workflow level, similar to application level. Thus, it deals with the failures of the execution of sub-workflows or workflow fragments by resubmitting the affected fragment.
—Task level: At this level, task failures are detected after perceiving failures and then being resolved.

Proactive Fault-Tolerance. This technique avoids waiting until the failures or errors occur and then recovering from them by foreseeing the failure and proactively substituting those components that have been suspected from those other components that are working properly [77]. It can resolve the faults at either:

—Workflow level: The aim here is to predict the failures at workflow level; in other words, it predicts the failures of sub-workflows or workflow fragments. For example, if an error is predicted for a given workflow fragment, it is replaced proactively from other working fragments.
—Task level: The focus here is on the tasks of workflow, where it predicts task failures and replaces them proactively from other working tasks.

6.13 Workflow Security

In the context of big data workflow, securing big data is not the whole story. It is considered as a part in preserving workflow security. The other part is guaranteeing the security of workflow logic and computation. Workflow security aims to secure the data-intensive tasks, which process and generate vast amounts of data. It is intended to ensure the security and integrity of the logic of operations. As big data workflow is an emerging research topic, there is a very limited research on this problem; therefore, preserving the security of this type of workflow is still an open issue (see Section 8). However, some existing techniques that can be utilized are Multicloud architectures, replication-based techniques (with or without a trust component called verifier), and task selection techniques [15, 113]. In addition, there are other research works that have proposed Multicloud security framework for data intensive workflows such as Ref. [32] and the secure workflow deployment technique on federated clouds such as in Ref. [135].

Table 2 shows the example work(s) for each classification level (approach) in the presented research taxonomy.

7 SYSTEMS WITH BIG DATA WORKFLOW SUPPORT

There are several platforms that can be extended or have the capability to support big data workflows. As discussed before, scientific workflow management systems also require coping with large volumes of data; hence, in recent years, many of them have been extended to support big data applications. Similarly, there are platforms/systems that have been designed specifically for orchestrating the execution of big data applications such as YARN. In this section, we survey these systems, respectively.

7.1 Scientific Workflow Systems with Big Data Extensions

Since the demand for data-intensive scientific workflow has increased and with the emerging of big data technology, several research works have extended the functionalities of existing SWMSs with data-intensive capabilities in order to enable big data applications in SWMS. We summarize and compare those research works (a.k.a. data-intensive scientific workflow management systems) in Table 4 of Appendix B.

7.2 Big Data Application Orchestrator

There are existing workflow tools that can be integrated with Hadoop to support MapReduce workflows, which are Luigi, Linkedin Azkaban, Apache Oozie, and Airflow. These tools are specific-purpose workflow managers that do not need to support the dynamism and heterogeneity of big data workflows. Rachel Kempf [81] compared these tools and highlighted their features. In the same context, Garg et al. [44] reviewed and compared the current orchestration tools for big data. Some tools reviewed in this research book chapter are also specific-purpose workflow managers. Accordingly, there are mainly three big data orchestrating systems that can be extended for big data workflow management. They are Apache Yet Another Resource Negotiator (YARN), Apache Mesos, and Amazon Lambda. The details of each of these platforms can be studied from Appendix C. As these platforms can be extended for big data workflow management, we discuss their capabilities against the challenges taxonomy that are shown in Figure 2.

Cloud Resource Management Challenge. For Apache YARN and Apache Mesos, the number and types of compute resources that will be allocated to workflow tasks need to be pre-selected as well as the configuration of these resources being determined. Therefore, these systems use limited resource provisioning since the provisioned resources are pre-determined and limited during the workflow execution (the only available resources for this workflow). Of course, both of them take into consideration managing all the available compute resources in all machines in the managed cluster. For AWS Lambda, the compute resource is determined based on the amount of memory, so that the amount of memory allocated to the Lambda function needs to be pre-determined. AWS Lambda allocates the power of CPU proportional to that amount by using the same ratio as a general purpose Amazon EC2 instance type [8]. The scaling of compute capacity is done dynamically by AWS Lambda in accordance to traffic load [8].

Table 2. Exemplar Works under Each Classification Level in the Presented Research Taxonomy

Taxonomy	Approaches	Exemplar Work
Cloud Platform Integration	Standardization Approach	Standardization bodies: NIST, DMTF, SNIA and ITU-T [98] Major open standards: OVF (by DMTF), CDMI (by SNIA) and OCCI (through OGF) [142]
	Intermediation Approach	SimpleCloud, DeltaCloud, Libcloud, jClouds and mOSAIC
Cross-Cloud Workflow Migration	Workflow Abstraction-based Approach	Makeflow [6], Asterism DIaaS [38] and dispel4py [39, 40]
	Cloud Broker-based Approach	Cloud service broker [65], STRATOS [97] and Broker-based framework for workflow applications [66]
	Container-based Approach	Skyport [45], Containerization strategies within workflow system [149], TOSCA-based platform [101], Asterism [38] and CoESMS [68]
Resource Provisioning	Static Resource Provisioning	AROMA [73]
	Dynamic Resource Provisioning	Cost-aware and SLA-based algorithms [7], RPS [19], Resource provisioning method [64] and Data-aware provisioning algorithm [122]
Resource Volatility	VM-level Mitigation Approach	COLO [33], VM workload consolidation-based fault-tolerance technique [74] and Hybrid adaptive checkpointing technique [115]
	Big Data Processing Framework-level Mitigation Approach	MapReduce Online [26], BFT MapReduce technique and prototype [28], Falkirk wheel [62], Mapreduce replication-based fault-tolerance technique [80] and Checkpointing & confined and replica recovery techniques for dataflow systems [138]
	Workflow Task-level Mitigation Approach	Workflow task-level techniques in workflow fault-tolerance classification can be used here
Data Storage	Cloud Storage	Amazon Cloud Storage (S3) , Microsoft Azure Data Storage , Google Cloud Storage , Rackspace Database Service and Rackspace Cloud Block Storage
	Shared Cloud-backed file system	Gfarm [90], BlueSky [131], DepSky [11], WaFS [133], Týr [94] and Faodel [124]
Data Movement	Data Transfer	Online parallel compression framework [14] and Data throttling technique in the proposed system [92]
	Data Locality	Two-stage data placement method [144], Task placement method [35], Heuristic data placement method [147], Clustering method based task dependency [92], GEODIS [27] and Fair job scheduler [21]
	Co-location Data and Computation	DPPACS [106], Task assignment method [148] and DACS [52, Chapter 18]
Data Provenance	Coarse-grained Provenance	Stream provenance method [128] and Workflow provenance management in WorkflowDSL [37]
	Fine-grained Provenance	RAMP [96], On-the-fly provenance tracking technique [111], Ariadne [48], Big data provenance techniques [23], Titian [60, 61] and DfAnalyzer [114]
Data Indexing	Traditional / Non-AI-based Indexing	Bitmap [136], Diff-Index [121], Inverted index pruning approach [129], UQE-Index [84], GIN , Metadata index and search system [140], SpatialHadoop [36]
	AI-based Indexing	GRAIL [139] and Semantic indexing technique [108]
	Collaborative AI-based Indexing	Collaborative semantic technique [42], Collaborative learning [41] and Collaborative filtering technique [71]
Workflow Specification Language	Generic Approach	YAWL [125] and CWL [9]
	Custom Approach	WorkflowDSL language [37]
Workflow Initialization	Non-Constraint-based Partitioning	Workflow partitioning method [25]
	Constraint-based Partitioning	Workflow partitioning based on storage constraints [24], PDWA [4] and I-PDWA [5]
Workflow Parallelization	Coarse-grained Parallelization	Workflow-level parallelism in Globus Genomics system [13] and Type-A workflow execution algorithm [91]
	Fine-grained Parallelization	Online parallel compression framework [14], Type-B workflow execution algorithm [91]
Workflow Scheduling	Push-based Scheduling	SLA-Based resource scheduling [146], Dynamic fault-tolerant scheduling method [151], Tree-to-tree task scheduling technique [148], Stable online scheduling strategy [119], T-Cluster algorithm [91], Elastic online scheduling [118], GEODIS [27] and Fair job scheduler [21]
	Pull-based Scheduling	Data-aware work stealing technique [132, 134]
Workflow Fault-Tolerance	Reactive Fault-Tolerance	Fault-tolerance scheduling algorithm [99], Fault-tolerant scheduling technique (FASTER) [151] and Fault-tolerance scheduling heuristics [100]
	Proactive Fault-Tolerance	FTDG [117]

Data Management and Storage Challenge. In Apache YARN, the use of HDFS is to store large amounts of data on cheap clusters and to provide high-performance access to that data across the cluster. Thus, Apache YARN utilizes a cloud-backed file system approach that allows it to deploy a shared file system in the cloud for a workflow. Moreover, for data movement, Apache YARN exploits data locality, and since the RM is a central authority and has a global view of cluster resources, it can enforce locality across tenants [126]. On the other hand, Apache Mesos offers persistent volumes to store data. The persistent volume can be created once a new task is launched, exists outside the sandbox of a task, provides exclusive access to a task by default, and will persist on the slave node even after the task finishes or dies. Shared persistent volumes are also supported by Apache Mesos in order to allow sharing of a volume between multiple tasks operating on the same node.

Data Security and Privacy Challenge. Authentication is supported by all systems. Where Apache YARN uses Kerberos authentication, Apache Mesos uses a factor authentication approach, i.e., a challenge-response protocol (CRAM-MD5), which is essentially single-factor authentication. AWS Lambda also supports a factor authentication approach, i.e., multi-factor authentication. In addition, controlling the access to resources and services is provided by the reviewed systems. Apache YARN supports coarse-grained access control, while Apache Mesos supports some extent fine-grained access control, and AWS Lambda provides fine-grained access control via AWS IAM. Moreover, to encrypt data and for communication data to remain private and integral, Apache YARN and Apache Mesos use Secure Sockets Layer (SSL) and Hypertext Transfer Protocol Secure (HTTPS), where SSL uses public-key cryptography at first and then symmetric cryptography for the rest of the computation to encrypt the transmitted data.

Workflow Scheduling Challenge. Apache YARN offers a single centralized scheduler to schedule competing workflow tasks among compute resources in the cluster. Therefore, Apache YARN uses a push-based approach with static scheduling. On the other hand, in Apache Mesos, the distributed two-level scheduling mechanism lets this framework either accept the offer or reject it. After the offered resource is accepted by the framework, this framework passes the task description to Apache Mesos, which launches the tasks on the corresponding agents [116] using a push-based approach. This scheduling mechanism is called "resource offers."

Workflow Fault-Tolerance Challenge. In Apache YARN, the RM detects and recovers from its own failures, where with work-preserving RM restart, the running applications will not lose their works, as well as RM detecting the failures of NM and AM and recovering them. In addition, in Apache Mesos, the failure of the master is detected and automatically recovered, where the running tasks can continue to execute in the case of failover [83]. Accordingly, the tasks of a workflow will not be affected by the failure of RM (with work-preserving RM restart) in Apache YARN or the failure of the master in Apache Mesos. However, the responsibility of handling the failures of containers in Apache YARN is by frameworks themselves [126] as well as the failures of node and executor in Apache Mesos being reported to framework schedulers and letting them take the appropriate actions to react to these failures, so that the responsibility of recovering from failures is by frameworks themselves. According to that, Apache YARN and Apache Mesos do not provide a mechanism to handle failures at application/framework level. As a result, Apache YARN and Apache Mesos use a reactive fault-tolerance approach for detecting and recovering from the failures of their masters. They have no mechanism for handling the failures at the level of workflow application and leave this responsibility to the workflow application itself, which reacts with its failures that may occur. For AWS Lambda as a serverless compute service, the underlying infrastructure is automatically managed, and in the cases where the Lambda function fails during processing an event, the functions invoked synchronously will reply with an exception and the functions invoked asynchronously are retried at least three times. If the input streams of the Lambda function come from Amazon Kinesis streams and Amazon DynamoDB streams, these streams/events are retried until this function succeeds or the data expires, where the data remains in Amazon Kinesis streams and Amazon DynamoDB streams for at least 24 hours [8]. Thus, it does not provide a mechanism to handle application-level failure, so that the fault-tolerance mechanism for workflow tasks is the responsibility of the workflow application.

8 OPEN ISSUES

In previous sections, several research studies have been highlighted that addressed big data workflow challenges and issues. Despite these efforts, some challenges are still open and not yet resolved, and others have not yet been investigated. In this section, we discuss key open research issues for big data workflow orchestration.

(1) *Workflow Interoperability and Openness*—Since the execution of big data workflow is carried out in the cloud, there is an opportunity to achieve the level of interoperability between cloud-based workflow systems via standard models for interoperability and cooperation. Thus, the integrated execution of big data workflows from heterogeneous workflow systems and different cloud platforms is needed. It allows workflow reuse and automation, enables workflow sharing, and workflow migration.

(2) *Workflow Fault-Tolerance and Dependability*—Several techniques and mechanisms of workflow fault-tolerance have been proposed to handle failures occurring during workflow execution and ensure its availability and reliability; however, supporting dependable big data workflow is still a complex task. The dynamism of such workflows and execution environments as well as the lengthy execution process are all factors that need to be considered. Generally, handling the failures that occurred requires first catching the error, identifying its source, then reducing its impact, and finally taking the appropriate actions to recover from it. Considering a "Cloud of clouds" environment, achieving those tasks is even harder due not only to the characteristics of big data and big data workflow, but also because of the characteristics of such environments.

(3) *Distributed Workflow Execution*—The dynamism of big data workflow due to data coming in different formats, velocity, and volumes [150] poses the need for distributed execution of such workflow over clouds to gain the benefits of both parallel data processing and the dynamic nature of the execution environment, achieving data processing efficiencies and better performance. The Multicloud or "Cloud of clouds" architecture as an execution environment relies on multiple clouds to make such distributed execution possible. However, such architecture allows avoiding vendor look-in and provides more flexibility on the one hand, and on the other hand, it complicates the whole execution process and related processes such as scheduling and parallelization, resulting in challenges and issues still being open, such as balancing workloads among clouds or reducing the cost of moving large datasets between workflow tasks/fragments.

(4) *Workflow Security*—Despite the benefits gained from using cloud computing and big data processing platforms, establishing standardized holistic solutions to security and privacy issues associated with moving big data workflow applications and their data to the cloud are still an important open issue. Comprehensive security solutions need to integrate the security of data-intensive tasks involved in workflow applications with the security of the consumed, generated, and produced big data. The industry is further challenged by the regulatory requirements that are different in each jurisdiction, with a trend to become increasingly protective and prescriptive, as in the general data protection regulation in the

European Union (regulation "EU 2016/679"). Novel technologies, such as blockchain, provide potential solutions for trusted cloud provision of computational services, but at the same time, pose new challenges with respect to privacy and scalability. Although many point solutions exist for security, and trusted platforms have been proposed (at operating system as well as application integration levels), the above-mentioned increasingly challenging environment presents a need to expand on this through research in new security and privacy platforms for the ultra-dynamic environment of emerging big data workflows. Solutions may often not be technological only, but marry economic, business, or personal incentives of stakeholders with the opportunities provided by technologies (see Ref. [34] for an example), thus providing solutions that are not only technically feasible but also leveraged and aligned with stakeholder interests.

(5) *User Perspective*—Despite the necessity of achieving the requirements of orchestrating big data workflow in the cloud, the requirements of users for the workflow should also be considered and accomplished. Thus, various requirements and constraints from different users result in different steps of a workflow needing to be executed, where the execution of these different steps might not be straightforward as the requirements may be conflicted. To clarify that, let us consider a data pipeline example in transportation, which is a workflow for analyzing traffic flow on the roads. The driver and traffic red light management are examples of users for this workflow and these users define performance requirements as SLA requirements but from different contexts. For the driver, it would be getting the analyzed results for congestion on the road quickly, allowing him/her to slow down the car speed before this congestion; and for traffic red light management, it would be getting the analytical results on the density of roads and traffic volume changes quickly, allowing it to react accordingly to avoid any congestion that could happen.

(6) *Cross-Cloud Workflow Migration Management*—The important open research issue relating to workflow migration management is twofold: (1) finding the equivalent instances in the target cloud environment since the exact equivalent for instances between original cloud and target cloud may not exist, and (2) transferring large datasets to the target cloud environment in case of such data stored in the original cloud. These issues complicate the workflow migration task, and along with the absence of universally accepted standards that make uniform the communication with the cloud, and provisioning and managing cloud resources [67], it poses the need to deal with vendor-specific platform at the target cloud.

(7) *Workflow Resources Operability and Volatility*—With different clouds, creating and registering virtual machine images for cloud resources differs. The open issue here is selecting or customizing images offered by cloud providers in order to achieve different requirements of orchestrating big data workflow in the cloud. For example, different tasks of a workflow may require different software stacks to run, which means different images are required. Moreover, virtual resources may be provisioned from different clouds, so that maintaining and tracking these resources during the whole execution of big data workflow is a difficult issue since those resources are distributed and reside in various cloud platforms, and are provisioned and released on demand.

Although cloud computing provides cloud resources on demand, the dynamic nature of cloud resources poses the need to deal with their volatilities because the loss of those resources often happens as a consequence of different failures [67]. That is crucial for big data workflow since the execution of such workflow is usually a lengthy process. Therefore, the configuration of virtual machines required for running workflow tasks, the new data products attached to virtual machines, and the intermediate and output big data products generated must all be stored and maintained during the whole execution of workflow to avoid any unexpected losses due to the loss of resources, whether they were virtual machines and/or storage volumes.

(8) *Cross-Layer Resources Configuration Selection*—With different software-based data processing primitives (such as batch processing or stream processing) that are implemented by different PaaS-level big data programming frameworks on IaaS-level resources, there is a need for cross-layer resource configuration selection techniques. The open issue here is to automatically select the configurations for both IaaS-level resource and PaaS-level framework to consistently accomplish the anticipated workflow-level SLA requirements, while maximizing the utilization of cloud datacenter resources [103].

9 CONCLUSION

Big Data Workflows consists of cross-disciplinary applications where timely results are critical: agriculture, transport, water management, healthcare, finance, utility networks, and environmental monitoring. However, exploiting the benefits of such big data workflows requires a deep and fundamental understanding of the different software, hardware, tools, and techniques required to compose, choreograph, and orchestrate such new types of workflows.

In this article, we outlined requirements for big data workflows in the cloud, presented a research taxonomy, and reviewed the approaches and techniques available for orchestrating big data workflow in the cloud. We also reviewed big data workflow systems and compared them against the presented research taxonomy. In addition, we discussed research problems that are still open for future research.

ACKNOWLEDGMENTS

This research is supported by an Australian Government Research Training Program (RTP) Scholarship. This research is also partially funded by two Natural Environment Research Council (UK) projects including (LANDSLIP: NE/P000681/1 and FloodPrep: NE/P017134/1).

REFERENCES

[1] [n.d.]. Chapter 15 - A taxonomy and survey of fault-tolerant workflow manag. sys. in cloud and dist. computing env. In *Software Architecture for Big Data and the Cloud*, Ivan Mistrik, Rami Bahsoon, Nour Ali, Maritta Heisel, and Bruce Maxim (Eds.). Morgan Kaufmann.

[2] 2015. Anomaly Detection over Sensor Data Streams. Retrieved from http://wiki.clommunity-project.eu/pilots:and.

[3] Adamu et al. 2016. *A Survey on Big Data Indexing Strategies.* Technical Report. SLAC National Accelerator Lab., Menlo Park, CA.

[4] Ahmad et al. 2014. Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems. In *Proceedings of the 4th International Conference on Big Data and Cloud Computing (BdCloud).* IEEE, 129–136.

[5] Ahmad et al. 2017. Optim. of data-intensive workflows in stream-based data process. models. *J Supercomput.* 73, 9 (2017), 3901–3923.

[6] Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain. 2012. Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies.*

[7] Alrokayan et al. 2014. Sla-aware provisioning and scheduling of cloud resources for big data analytics. In *CCEM.* IEEE, 1–8.

[8] Amazon. 2017. AWS Lambda. Retrieved from https://aws.amazon.com/lambda/details/.

[9] Amstutz et al. 2016. Common workflow language, draft 3.

[10] Beloglazov et al. 2012. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.* 28, 5 (2012), 755–768.

[11] Bessani et al. 2013. DepSky: Dependable and secure storage in a cloud-of-clouds. *ACM Trans. Storage (TOS)* 9, 4 (2013), 12.

[12] Bessani et al. 2014. SCFS: A shared cloud-backed file system. In *USENIX Annual Technical Conference.*

[13] Bhuvaneshwar et al. 2015. A case study for cloud based high throughput analysis of NGS data using the globus genomics system. *Comput. Struct. Biotechnology J.* 13 (2015), 64–74.

[14] Bicer et al. 2013. Integrating online compression to accelerate large-scale data analytics applications. In *Proceedings of the 27th International Symposium on Parallel & Distributed Processing (IPDPS).* IEEE, 1205–1216.

[15] Bohli et al. 2013. Security and privacy-enhancing multicloud arch. *IEEE Trans. Dependable Secure Comput.* 10, 4 (2013), 212–224.

[16] Marc Bux and Ulf Leser. 2013. Parallelization in scientific workflow management systems. *arXiv preprint arXiv:1303.7195* (2013).

[17] Massimo Cafaro and Giovanni Aloisio. 2011. Grids, clouds, and virtualization. In *Grids, Clouds and Virtualization.* Springer, 1–21.

[18] Cai et al. 2017. IoT-based big data storage systems in cloud comp.: Perspectives and challenges. *IEEE IoT J.* 4, 1 (2017), 75–87.

[19] Cao et al. 2016. A resource provisioning strategy for elastic analytical workflows in the cloud. In *Proceedings of the 18th International Conference on High-Performance Computing and Communications, 14th International Conference on Smart City, and 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS).* IEEE, 538–545.

[20] Chen et al. 2013. Big data challenge: A data management perspective. *Front. Comput. Sci.* 7, 2 (2013), 157–164.

[21] Chen et al. 2018. Scheduling jobs across geo-distributed datacenters with max-min fairness. *IEEE Trans. Network Sci.Eng.* (2018). PrePrints. DOI: 10.1109/TNSE.2018.2795580

[22] CL Philip Chen and Chun-Yang Zhang. 2014. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Inf. Sci.* 275 (2014), 314–347.

[23] Peng Chen. 2016. Big data analytics in static and streaming provenance.

[24] Weiwei Chen and Ewa Deelman. 2011. Partitioning and scheduling workflows across multiple sites with storage constraints. In *Proceedings of the International Conference on Parallel Processing and Applied Mathematics.* Springer.

[25] Weiwei Chen and Ewa Deelman. 2012. Integration of workflow partitioning and resource provisioning. In *Proceedings of the 12th International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012).* IEEE Computer Society, 764–768.

[26] Condie et al. 2010. MapReduce online. In *NSDI*, Vol. 10. 20.

[27] Convolbo et al. 2018. GEODIS: Towards optim. of data locality-aware job sched. in geo-distrib. datacenters. *Comput.* 100, 1 (2018), 21–46.

[28] Costa et al. 2011. Byzantine fault-tolerant MapReduce: Faults are not just crashes. In *Proceedings of the 3rd International Conference on Cloud Computing Technology and Science (CloudCom).* IEEE, 32–39.

[29] Costa et al. 2014. Towards an adaptive and distributed architecture for managing workflow provenance data. In *Proceedings of the 10th International Conference on e-Science (e-Science)*, Vol. 2. IEEE.

[30] Alfredo Cuzzocrea. 2014. Privacy and security of big data: Current challenges and future research perspectives. In *Proceedings of the 1st International Workshop on Privacy and Secuirty of Big Data.* ACM.

[31] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[32] Demchenko et al. 2017. Defining intercloud security framework and architecture components for multi-cloud data intensive applications. In *Proceedings of the 17th International Symposium on Cluster, Cloud and Grid Computing.* IEEE Press, 945–952.

[33] Dong et al. 2013. COLO: COarse-grained LOck-stepping virtual machines for non-stop service. In *Proceedings of the 4th Annual Symposium on Cloud Computing.*

[34] Dong et al. 2017. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *Proceedings of the SIGSAC Conference on Computer and Communications Security.* ACM, 211–227.

[35] Ebrahimi et al. 2015. TPS: A task placement strategy for big data workflows. In *Proceedings of the International Conference on Big Data (Big Data).* IEEE, 523–530.

[36] Ahmed Eldawy and Mohamed F. Mokbel. 2015. Spatialhadoop: A mapreduce framework for spatial data. In *Proceedings of the IEEE 31st International Conference on Data Engineering (ICDE'15).* IEEE, 1352–1363.

[37] Fernando et al. 2018. WorkflowDSL: Scalable workflow execution with provenance for data analysis applications. In *Proceedings of the 42nd Annual Computer Software and Applications Conference (COMPSAC).* IEEE, 774–779.

[38] Filgueira et al. 2016. Asterism: Pegasus and dispel4py hybrid workflows for data-intensive science. In *Proceedings of the 7th International Workshop on Data-Intensive Computing in the Cloud.* IEEE Press.

[39] Rosa Filgueira, Amrey Krause, Malcolm Atkinson, Iraklis Klampanos, Alessandro Spinuso, and Susana Sanchez-Exposito. 2015. dispel4py: An agile framework for data-intensive escience. In *Proceedings of the IEEE 11th International Conference on e-Science (e-Science'15).* IEEE, 454–464.

[40] Rosa Filguiera, Amrey Krause, Malcolm Atkinson, Iraklis Klampanos, and Alexander Moreno. 2017. dispel4py: A Python framework for data-intensive scientific computing. *Int. J. High Perform. Comput. Appl.* 31, 4 (2017), 316–334.

[41] Wai-Tat Fu and Wei Dong. 2012. Collabor. indexing and knowledge explor.: A social learn. model. *IEEE Intell. Syst.* 27, 1 (2012), 39–46.
[42] Gacto et al. 2010. Integration of an index to preserve the semantic interpretability in the multiobjective evolutionary rule selection and tuning of linguistic fuzzy systems. *IEEE Trans. Fuzzy Syst.* 18, 3 (2010), 515–531.
[43] Gani et al. 2016. A survey on indexing techniques for big data: Taxonomy and performance evaluation. *Knowl. Inf. Syst.* 46, 2 (2016), 241–284.
[44] Garg et al. 2018. *Orchestration Tools for Big Data.* Springer International Publishing, 1–9.
[45] Gerlach et al. 2014. Skyport: Container-based execution environment management for multi-cloud scientific workflows. In *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds.* IEEE Press, 25–32.
[46] George M. Giaglis. 2001. A taxonomy of business process modeling and information systems modeling techniques. *Int. J. Flexible Manuf. Syst.* 13, 2 (2001), 209–228.
[47] Glavic et al. 2011. The case for fine-grained stream provenance. In *BTW Workshops*, Vol. 11.
[48] Glavic et al. 2014. Efficient stream provenance via operator instrumentation. *ACM Trans. Internet Technol. (TOIT)* 14, 1 (2014), 7.
[49] Boris Glavic. 2014. Big data provenance: Challenges and implications for benchmarking. In *Specifying Big Data Benchmarks.* Springer, 72–80.
[50] Gomes et al. 2018. Enabling rootless Linux containers in multi-user envin.: The udocker tool. *Computer Physics Communications* (2018).
[51] Gonidis et al. 2013. Cloud application portability: An initial view. In *Proceedings of the 6th Balkan Conference in Informatics.* ACM.
[52] Hassan et al. 2017. *Networks of the Future: Architectures, Technologies, and Implementations.* Chapman and Hall/CRC.
[53] He et al. 2016. Efficient and anonymous mobile user authentication protocol using self-certified public key cryptography for multi-server architectures. *IEEE Trans. Inf. Forensics Secur.* 11, 9 (2016), 2052–2064.
[54] He et al. 2018. A provably-secure cross-domain handshake scheme with symptoms-matching for mobile healthcare social network. *IEEE Trans. Dependable and Secure Comput.* 15, 4 (2018), 633–645.
[55] Hirzel et al. 2013. IBM streams processing language: Analyzing big data in motion. *IBM J. Res. Dev.* 57, 3/4 (2013).
[56] Hu et al. 2014. Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access* 2 (2014), 652–687.
[57] Hu et al. 2016. Flutter: Scheduling tasks closer to data across geo-distributed datacenters. In *Proceedings of the 35th Annual IEEE INFOCOM.* 1–9.
[58] Hung et al. 2015. Scheduling jobs across geo-distributed datacenters. In *Proceedings of the 6th Symposium on Cloud Computing.* ACM, 111–124.
[59] Huq et al. 2011. Inferring fine-grained data provenance in stream data processing: Reduced storage cost, high accuracy. In *Proceedings of the International Conference on Database and Expert Systems Applications.* Springer.
[60] Interlandi et al. 2017. Adding data provenance support to Apache Spark. *The VLDB J.* (2017), 1–21.
[61] Matteo Interlandi and Tyson Condie. 2018. Supporting data provenance in data-intensive scalable comp. sys. *Data Eng.* (2018), 63.
[62] Michael Isard and Martín Abadi. 2015. Falkirk wheel: Rollback recovery for dataflow systems. *arXiv preprint arXiv:1503.08877* (2015).
[63] Jin et al. 2016. Workload-aware scheduling across geo-distributed data centers. In *Trustcom/BigDataSE/ISPA.* IEEE, 1455–1462.
[64] Todd Jr. et al. 2017. Data analytics computing resource provisioning based on computed cost and time parameters for proposed computing resource configurations. US Patent 9,684,866.
[65] Jrad et al. 2012. SLA based service brokering in intercloud environments. *CLOSER* 2012 (2012), 76–81.
[66] Jrad et al. 2013. A broker-based framework for multi-cloud workflows. In *Proceedings of the Intern. Workshop on Multi-cloud Applications and Federated Clouds.*
[67] Andrey Kashlev and Shiyong Lu. 2014. A system architecture for running big data workflows in the cloud. In *Proceedings of the International Conference on Services Computing (SCC).* IEEE, 51–58.
[68] Kaur et al. 2017. Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. *IEEE Wireless Commun.* 24, 3 (2017), 48–56.
[69] Tyler Keenan. 2016. Streaming Data: Big Data at High Velocity. Retrieved from https://www.upwork.com/hiring/data/streaming-data-high-velocity/.
[70] Kiran et al. 2015. Lambda architecture for cost-effective batch and speed bigdata process. In *Proceedings of the International Conference on Big Data.*
[71] Komkhao et al. 2013. Incremental collaborative filtering based on Mahalanobis distance and fuzzy membership for recommender systems. *Int. J. Gen. Syst.* 42, 1 (2013), 41–66.
[72] Kurtzer et al. 2017. Singularity: Scientific containers for mobility of compute. *PloS One* 12, 5 (2017), e0177459.

[73] Palden Lama and Xiaobo Zhou. 2012. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In *Proceedings of the 9th International Conference on Autonomic Computing*. ACM, 63–72.

[74] Li et al. 2017. Study on fault tolerance method in cloud platform based on workload consolidation model of virtual machine. *J. Eng. Sci. Technol. Rev.* 10, 5 (2017), 41–49.

[75] Lin et al. 2016. StreamScope: Continuous reliable distributed processing of big data streams. In *NSDI*. 439–453.

[76] Liu et al. 2014. Scientific workflow partitioning in multisite cloud. In *Proceedings of the European Conference on Parallel Processing*. Springer, 105–116.

[77] Liu et al. 2015. A survey of data-intensive scientific workflow management. *J. Grid Comput.* 13, 4 (2015), 457–493.

[78] Liu et al. 2016. Rethinking big data: A review on the data quality and usage issues. *ISPRS J. PRS* 115 (2016), 134–142.

[79] Liu et al. 2018. A survey of scheduling frameworks in big data systems. *Int. J. Cloud Comput.* (2018), 1–27.

[80] Yang Liu and Wei Wei. 2015. A replication-based mechanism for fault tolerance in mapreduce framework. *Math. Prob. Eng.* 2015 (2015).

[81] Rache lKempf. 2017. Open Source Data Pipeline—Luigi vs Azkaban vs Oozie vs Airflow. Retrieved from https://www.bizety.com/2017/06/05/open-source-data-pipeline-luigi-vs-azkaban-vs-oozie-vs-airflow/.

[82] Lopez et al. 2016. A performance comparison of Open-Source stream processing platforms. In *Proceedings of the Global Communications Conference (GLOBECOM)*.

[83] Dan Lynn. 2016. Apache Spark Cluster Managers: YARN, Mesos, or Standalone? Retrieved from http://www.agildata.com/apache-spark-cluster-managers-yarn-mesos-or-standalone/.

[84] Ma et al. 2012. An efficient index for massive IOT data in cloud environment. In *Proceedings of the 21st International Conference on IKM*. 2129–2133.

[85] Mace et al. 2011. The case for dynamic security solutions in public cloud workflow deployments. In *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 111–116.

[86] Malik et al. 2010. Tracking and sketching distributed data provenance. In *Proceedings of the 6th International Conference on e-Science*. IEEE.

[87] Mansouri et al. 2017. Data storage management in cloud envirn.: Taxonomy, survey, and future directions. *ACM CSUR* 50, 6 (2017), 1–51.

[88] Di Martino et al. 2015. Cross-platform cloud APIs. In *Cloud Portability and Interoperability*. Springer, 45–57.

[89] Ulf Mattsson. 2016. Data centric security key to cloud and digital business. Retrieved from https://www.helpnetsecurity.com/2016/03/22/data-centric-security/.

[90] Mikami et al. 2011. Using the Gfarm file system as a POSIX compatible storage platform for Hadoop MapReduce applications. In *Proceedings of the12th IEEE/ACM International Conference on Grid Computing (GRID)*. IEEE, 181–189.

[91] Mohan et al. 2016. A NOSQL data model for scalable big data workflow execution. In *Proceedings of the International Congress on Big Data (BigData Congress)*.

[92] Mon et al. 2016. Clustering based on task dependency for data-intensive workflow scheduling optimization. In *Proceedings of the 9th Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS)*. IEEE, 20–25.

[93] Nachiappan et al. 2017. Cloud storage reliability for big data applications: A state of the art survey. *J. Netw. Comput. Appl.* 97 (2017), 35–47.

[94] Matri et al. 2016. *Tỳr: Efficient Transactional Storage for Data-Intensive Applications*. Ph.D. Dissertation. Inria Rennes Bretagne Atlantique; Universidad Politécnica de Madrid.

[95] Suraj Pandey and Rajkumar Buyya. 2012. A survey of scheduling and management techniques for data-intensive application workflows. In *Data Intensive Distributed Computing: Challenges and Solutions for Large-scale Information Management*. IGI Global, 156–176.

[96] Park et al. 2011. Ramp: A system for capturing and tracing provenance in mapreduce workflows. In *Proceedings of 37th International Conference on Very Large Data Bases (VLDB'11)*.

[97] Pawluk et al. 2012. Introducing STRATOS: A cloud broker service. In *Proceedings of the 5th International Conference on Cloud Computing (CLOUD)*.

[98] Peoples et al. 2013. The standardisation of cloud computing: Trends in the state-of-the-art and management issues for the next generation of cloud. In *Proceedings of the Science and Information Conference (SAI)*. IEEE.

[99] Poola et al. 2014. Fault-tolerant workflow scheduling using spot instances on clouds. *Procedia Comput. Sci.* 29 (2014), 523–533.

[100] Poola et al. 2016. Enhancing reliability of workflow execution using task replication and spot instances. *ACM Trans. Auton. Adapt. Syst. (TAAS)* 10, 4 (2016), 1–30.

[101] Qasha et al. 2016. Dynamic deployment of scientific workflows in the cloud using container virtualization. In *Proceedings of the International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 269–276.

[102] Rahman et al. 2011. A taxonomy and survey on autonomic management of applications in grid computing environments. *Concurrency Comput. Pract. Experience* 23, 16 (2011), 1990–2019.

[103] Ranjan et al. 2015. Cross-layer cloud resource configuration selection in the big data era. *IEEE Cloud Comput.* 2, 3 (2015), 16–22.

[104] Ranjan et al. 2017. Orchestrating BigData analysis workflows. *IEEE Cloud Comput.* 4, 3 (2017), 20–28.

[105] Rao et al. 2019. The big data system, components, tools, and technologies: A survey. *Knowl. Inf. Syst.* 60, 3 (2019), 1165–1245.

[106] K. H. K. Reddy and D. S. Roy. 2015. Dppacs: A novel data partitioning and placement aware computation scheduling scheme for data-intensive cloud applications. *Comput. J.* 59, 1 (2015), 64–82.

[107] Maria Alejandra Rodriguez and Rajkumar Buyya. 2017. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. *Concurrency Comput. Pract. Experience* 29, 8 (2017).

[108] Rodríguez-García et al. 2014. Creating a semantically-enhanced cloud services environment through ontology evolution. *Future Gener. Comput. Syst.* 32 (2014), 295–306.

[109] Sakr et al. 2011. A survey of large scale data management approaches in cloud envirns. *IEEE Commun. Surv. Tutorials* 13, 3 (2011), 311–336.

[110] Sakr et al. 2013. The family of mapreduce and large-scale data processing systems. *ACM Comput. Surv. (CSUR)* 46, 1 (2013), 11.

[111] Sansrimahachai et al. 2013. An on-the-fly provenance tracking mechanism for stream processing systems. In *Proceedings of the 12th International Conference on Computer and Information Science (ICIS)*. IEEE, 475–481.

[112] Seiger et al. 2018. Toward an execution system for self-healing workflows in cyber-physical systems. *Software & Syst. Model.* 17, 2 (2018), 551–572.

[113] Shishido et al. 2018. (WIP) tasks selection policies for securing sensitive data on workflow scheduling in clouds. In *IEEE SCC*.

[114] Silva et al. 2018. DfAnalyzer: Runtime dataflow analysis of scientific applications using provenance. *VLDB Endowment* 11, 12 (2018).

[115] Souza et al. 2018. Hybrid adaptive checkpointing for VM fault tolerance. In *Proceedings of the International Conference on Cloud Engineering (IC2E)*.

[116] Mesos Sphere. 2017. Apache Mesos. Retrieved from https://mesosphere.com/why-mesos/?utm_source=adwords&utm_medium=g&utm_campaign=43843512431&utm_term=mesos&utm_content=190805957225&gclid=CLqw8o6J6dMCFdkGKgodYlsD_A.

[117] Sun et al. 2017. Building a fault tolerant framework with deadline guarantee in big data stream computing environments. *J. Comput. Syst. Sci.* 89 (2017), 4–23.

[118] Sun et al. 2018. Rethinking elastic online scheduling of big data streaming applications over high-velocity continuous data streams. *J. Supercomputing* 74, 2 (2018), 615–636.

[119] Dawei Sun and Rui Huang. 2016. A stable online scheduling strategy for real-time stream computing over fluctuating big data streams. *IEEE Access* 4 (2016), 8593–8607.

[120] Talbi et al. 2012. Multi-objective optimization using metaheuristics: Non-standard algorithms. *Int. Trans. Oper. Res.* 19, 1-2 (2012), 283–305.

[121] Tan et al. 2014. Diff-Index: Differentiated index in distributed log-structured data stores. In *EDBT*. 700–711.

[122] Toosi et al. 2018. Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using Aneka. *Future Gener. Comput. Syst.* 79, 2 (2018), 765–775.

[123] Tudoran et al. 2016. Overflow: Multi-site aware big data management for scientific workflows on clouds. *IEEE TCC* 4, 1 (2016), 76–89.

[124] Ulmer et al. 2018. Faodel: Data management for next-generation application workflows. In *Proceedings of the 9th Workshop on Scientific Cloud Computing*.

[125] Wil M. P. Van Der Aalst and Arthur HM Ter Hofstede. 2005. YAWL: Yet another workflow language. *Inf. Syst.* 30, 4 (2005), 245–275.

[126] Vavilapalli et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*. ACM.

[127] Venkataraman et al. 2017. Drizzle: Fast and adaptable stream processing at scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 374–389.

[128] Nithya Vijayakumar and Beth Plale. 2007. Tracking stream provenance in complex event processing systems for workflow-driven computing. In *Proceedings of the EDA-PS Workshop*.

[129] Vishwakarma et al. 2014. An eff. approach for inverted index pruning based on document relevance. In *Proceedings of the 4th International Conference on CSNT*.

[130] von Leon et al. 2019. A lightweight container middleware for edge cloud architectures. *Fog and Edge Computing: Principles and Paradigms* (2019), 145–170.

[131] Vrable et al. 2012. BlueSky: A cloud-backed file system for the enterprise. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*.

[132] Wang et al. 2014. Optimizing load balancing and data-locality with data-aware scheduling. In *Proceedings of the International Conference on Big Data (Big Data)*.

[133] Wang et al. 2015. WaFS: A workflow-aware file system for effective storage utilization in the cloud. *IEEE Trans. Comput.* 64, 9 (2015), 2716–2729.

[134] Wang et al. 2016. Load-balanced and locality-aware scheduling for data-intensive workloads at extreme scales. *Concurrency Comput. Pract. Experience* 28, 1 (2016), 70–94.

[135] Wen et al. 2017. Cost effective, reliable and secure workflow deployment over federated clouds. *IEEE TSC.* 10, 6 (2017), 929–941.

[136] Wu et al. 2010. Analyses of multi-level and component compressed bitmap indexes. *ACM Trans. Database Syst.* 35, 1 (2010), 2.

[137] Wu et al. 2015. Workflow scheduling in cloud: A survey. *J. Supercomput.* 71, 9 (2015), 3373–3418.

[138] Xu et al. 2017. On fault tolerance for distributed iterative dataflow processing. *IEEE Trans. KDE* 29, 8 (2017), 1709–1722.

[139] Yıldırım et al. 2012. GRAIL: A scalable index for reachability queries in very large graphs. *VLDB J.* 21, 4 (2012), 509–534.

[140] Yu et al. 2014. An efficient multidimension metadata index and search system for cloud data. In *Proceedings of the 6th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 499–504.

[141] Jia Yu and Rajkumar Buyya. 2005. A taxonomy of scientific workflow systems for grid computing. *ACM Sigmod Record* 34, 3 (2005), 44–49.

[142] Zhang et al. 2013. A survey on cloud interoperability: taxon., stand., and practice. *ACM SIGMETRICS Perf. Eval. Rev.* 40, 4 (2013), 13–22.

[143] Zhao et al. 2014. Devising a cloud scientific workflow platform for big data. In *World Congress on Services (SERVICES)*. IEEE.

[144] Zhao et al. 2015. A data placement strategy for data-intensive scientific workflows in cloud. In *Proceedings of the 15th IEEE/ACM CCGRID*. 928–934.

[145] Zhao et al. 2015. Enabling scalable scientific workflow management in the Cloud. *Future Gener. Comput. Syst.* 46 (2015), 3–16.

[146] Zhao et al. 2015. SLA-based resource scheduling for big data analytics as a service in cloud computing environments. In *Proceedings of the 44th International Conference on Parallel Processing (ICPP)*. IEEE, 510–519.

[147] Zhao et al. 2016. Heuristic data placement for data-intensive applications in heterogeneous cloud. *JECE* (2016).

[148] Zhao et al. 2016. A new energy-aware task scheduling method for data-intensive applications in the cloud. *JNCA* 59 (2016), 14–27.

[149] Charles Zheng and Douglas Thain. 2015. Integrating containers into workflows: A case study using makeflow, work queue, and docker. In *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing*. ACM, 31–38.

[150] Chaochao Zhou and Saurabh Kumar Garg. 2015. Performance analysis of scheduling algorithms for dynamic workflow applications. In *Proceedings of the International Congress on Big Data (BigData Congress)*. IEEE.

[151] Zhu et al. 2016. Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. *IEEE Trans. Parallel Distrib. Syst.* 27, 12 (2016), 3501–3517.

Received October 2018; revised March 2019; accepted May 2019