Journal of Big Data

SURVEY PAPER Open Access

A survey on data storage and placement methodologies for Cloud-Big Data ecosystem

Somnath Mazumdar[1], Daniel Seybold[2], Kyriakos Kritikos[3*] and Yiannis Verginadis[4]

*Correspondence:
kritikos@ics.forth.gr
[3] ICS-FORTH, Heraklion, Crete, Greece
Full list of author information is available at the end of the article

Abstract
Currently, the data to be explored and exploited by computing systems increases at an exponential rate. The massive amount of data or so-called "Big Data" put pressure on existing technologies for providing scalable, fast and efficient support. Recent applications and the current user support from multi-domain computing, assisted in migrating from data-centric to knowledge-centric computing. However, it remains a challenge to optimally store and place or migrate such huge data sets across data centers (DCs). In particular, due to the frequent change of application and DC behaviour (i.e., resources or latencies), data access or usage patterns need to be analyzed as well. Primarily, the main objective is to find a better data storage location that improves the overall data placement cost as well as the application performance (such as throughput). In this survey paper, we are providing a state of the art overview of Cloud-centric Big Data placement together with the data storage methodologies. It is an attempt to highlight the actual correlation between these two in terms of better supporting Big Data management. Our focus is on management aspects which are seen under the prism of non-functional properties. In the end, the readers can appreciate the deep analysis of respective technologies related to the management of Big Data and be guided towards their selection in the context of satisfying their non-functional application requirements. Furthermore, challenges are supplied highlighting the current gaps in Big Data management marking down the way it needs to evolve in the near future.

Keywords: Big Data, Cloud, Data models, Data storage, Placement

Introduction

Over the time, the type of applications has evolved from batch, compute or memory intensive applications to streaming or even interactive applications. As a result, applications are getting more complex and become long-running. Such applications might require frequent-access to multiple distributed data sources. During application deployment and provisioning, the user can face various issues such as (i) where to effectively place both the data and the computation; (ii) how to achieve required objectives while reducing the overall application running cost. Data could be generated from various sources, including a multitude of devices over IoT environments that can generate a huge amount of data, while the applications are running. An application can further produce a large amount of data. In general, data of such size is usually referred to as *Big Data*. In general, Big Data is characterised by five properties [1, 2]. These are *volume*, *velocity* (means rapid update and propagation of data), *variety* (means different kinds of data parts), *veracity* (related to the trustworthiness, authenticity and protection (degree) of the data) and *value* (the main added-value and the importance of the data to the business). A large set of different data types generated from various sources can hold enormous information (in the form of relationships [3], system access logs, and also as the quality of services (QoSs)). Such knowledge can be critical for improving both products and services. Thus, to retrieve the underlying knowledge from such big sized data sets an efficient data processing ecosystem and knowledge filtering methodologies are needed.

In general, Cloud-based technology offers different solutions over different levels of abstractions to build and dynamically provision user applications. The Cloud offers suitable frameworks for the clustering of Big Data as well as efficiently distributed databases for their storage and placement. However, the native Cloud facilities have a lack of guidance on how to combine and integrate services in terms of holistic frameworks which could enable users to properly manage both their applications and the data. While there exist some promising efforts that fit well under the term *Big Data-as-a-service (BDaaS)*, most of them still lack adequate support for: data-privacy [4–6], query optimisation [7], robust data analytics [8] and data-related service level objective management for increased (Big Data) application quality [9]. Currently, the application placement and management over multi or cross-Clouds is being researched. However, the additional dimension of Big Data management does raise significantly the complexity of finding adequate and realistic solutions.

The primary goal of this survey is to present the current state-of-affairs in Cloud computing with respect to the Big Data management (mainly storage and placement) from the application's administration point-of-view. To this end, we have thoroughly reviewed the proposed solutions based on the placement and storage of Big Data through the use of a carefully designed set of criteria. Such criteria were devised under the prism of non-functional properties. This was performed in an attempt to unveil those solutions which can be deemed suitable for the better management of different kinds of applications (while taking into consideration non-functional aspects). In the end, the prospective readers (such as Big Data application owners, DevOps) can be guided towards the selection of those solutions in each Big Data management lifecycle phase (focused in this article) that satisfy in a better way their non-functional application requirements. The analysis finally concludes with the identification of certain gaps. Based on the latter, a set of challenges for the two Big Data management phases covered as well as for Big Data management as a whole are supplied towards assisting in the evolution of respective solutions and paving the way for the actual directions that the research should follow.

Based on the above analysis, it is clear that this article aims at providing guidance to potential adopters concerning the most appropriate solution for both placing and storing Big Data (according to the distinctive requirements of the application domain). To this end, our work can be considered as complementary to other relevant surveys that attempt to review Big Data technologies. In particular, the past surveys have focused on the deployment of data-intensive applications in the Cloud [10], on assessing various database management tools for storing Big Data [11], on evaluating the technologies developed for Big Data applications [12], on Cloud-centric distributed database management systems (primarily on NoSQL storage models) [13], on design principles for in-memory Big Data management and processing [14] and on research challenges related to Big Data in the Cloud ecosystem [15]. However, the primary focus of these surveys is mainly on functional aspects examined under the prism of analysing different dimensions and technology types related to Big Data. Further, there is no clear discussion on management aspects in the context of the whole Big Data management lifecycle as usually the focus seems to be merely on the Big Data storage phase. Interestingly, our survey deeply analyses those phases in the Big Data management lifecycle that are the most crucial in the context of satisfying application non-functional requirements.

The remaining part of this manuscript is structured as follows: "Data lifecycle management (DLM)" section explicates how data modelling can be performed, analyses various data management lifecycle models and comes up with an ideal one which is presented along with the proper architecture to support it. Next, "Methodology" section attempts to explain this survey's main methodology. "Non-functional data management features" section details the main non-functional features of focus in this article. Based on these features, the review of Big Data storage systems and distributed file systems are supplied in "Data storage systems" section. Similarly, the review of state-of-the-art data placement techniques is performed in "Data placement techniques" section. Next, "Lessons learned and future research directions" section presents relevant lessons learned as well as certain directions for future research work and finally "Concluding remarks" section concludes the survey paper.

Data lifecycle management (DLM)

Data lifecycle models

There exist two types of data lifecycle models focusing on either general data or Big Data management. The generic data management lifecycles usually cover activities such as generation, collection (curation), storage, publishing, discovery, processing and analysis of data [16].

In general, Big Data lifecycle models primarily comprises activities (such as data collection, data loading, data processing, data analysis and data visualisation [17, 18]). It is worth to note that apart from the data visualisation, they do share many identical activities with the generic ones. However, such models do not mention the value of data.

To counter this, the NIST reference model [19] suggests four data management phases: *collection*, *preparation*, *analysis* and *action*, where the *action* phase is related to using synthesised knowledge to create value (represents analytics and visualisation of knowledge). Furthermore, focusing more on the data value, OECD [20] has proposed a data value cycle model comprising six phases: *datafication and data collection*, *Big Data*, *data analytics*, *knowledge base*, *decision making* and *valued-added* for growth and well-being. The model forms an iterative, closed feedback loop where results from Big Data analytics are fed back to the respective database. Later, the work in [21] exposed the main drawbacks of OECD and proposed a new reference model that adds two additional components, the business intelligence (BI) system and the environment, into the OECD model. The data interaction and analysis formulates a short closed loop in the model. A greater loop is also endorsed via the BI's iterative interaction and observation of its environment. Finally, it is claimed that the management of Big Data for value creation is also linked to the BI management. In this way, Big Data management is related directly to the activities of data integration, analysis, interaction and effectuation along with the successful management of the emergent knowledge via data intelligence.

Data modelling

The data needs to be described in an appropriate form prior to any kind of usage. The information used for the data description is termed as metadata (i.e., data about data) [22–24]. The use of metadata enriches data management so that it can properly support and improve any data management activity. Two major issues related to metadata management are:

- *How should metadata be described (or characterised)?* The description of a metadata schema which can be exploited to efficiently place a certain Big Data application in multiple Clouds by respecting both user constraints and requirements. Such a metadata schema has been proposed partially in [25] or completely in [26].
- *How should metadata be efficiently managed and stored for better retrieval and exploitation?* The design of appropriate languages [27, 28] that focus on the description of how Big Data applications and data should be placed and migrated across different multiple Cloud resources.

For a better description of metadata, the authors in [22] identify available Cloud services and analyse some of their main characteristics following a tree-structured taxonomy. Another relevant effort is the DICE project [25] that focuses on the quality-driven development of Big Data applications. It offers a UML profile along with the appropriate tools that may assist software designers to reason about the reliability, safety and efficiency of data-intensive applications. Specifically, it has introduced a metamodel for describing certain aspects of Big Data-intensive applications.

Most of these efforts do not offer a direct support for expressing significant aspects of Big Data, such as data origin, location, volume, transfer rates or even aspects of the operations that transfer data between Cloud resources. One effort that tries to cover the requirements for a proper and complete metadata description is the Melodic metadata schema [26]. This schema refers to a taxonomy of concepts, properties and relationships that can be exploited for supporting Big Data management as well as application deployment reasoning. The schema is clustered into three parts: (i) one focusing on specifying Cloud service requirements and capabilities to support application deployment reasoning; (ii) another focusing on defining Big Data features and constraints to support Big Data management; (iii) a final one concentrating on supplying Big Data security-related concepts to drive the data access control.

With respect to the second direction of work, although several languages are currently used for capturing application placement and reconfiguration requirements (e.g., TOSCA [27]), a lack of distinct support for describing placement and management requirements for Big Data can be observed. However, if such languages are extended through the possible use of a metadata schema, then they could be able to achieve this purpose. This has been performed in [26], where a classical, state-of-the-art Cloud description language called CAMEL [29] has been extended to enable the description of Big Data placement and management requirements by following a feature-model-based

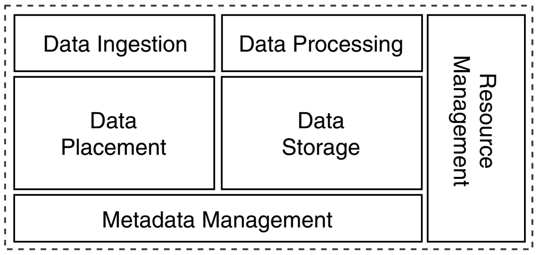

Fig. 1 A high-level block diagram of a Big Data management system

approach where requirements are expressed as features or attributes that are annotated via elements from the metadata schema.

Data lifecycle management systems

Traditional data lifecycle management systems (DLMSs) focus more on the way data is managed and not on how they are processed. In particular, the actual main services that they offer are data storage planning (and provisioning) and data placement (and execution support) via efficient data management policies. On the other hand, it seems that data processing is covered by other tools or systems as it is regarded as application-specific. Traditionally in Cloud, Big Data processing is offered as a separate service, while the resource management is usually handled by other tools, such as Apache Mesos or YARN. Figure 1 depicts the architecture of a system that completely addresses the data management lifecycle, as inscribed in the previous sub-section. This system comprises six primary components.

- *Metadata management* takes care of maintaining information which concerns both the static and dynamic characteristics of data. It is the cornerstone for enabling efficient data management.
- *Data placement* encapsulates the main methods for efficient data placement and data replication while satisfying user requirements.
- *Data storage* is responsible for proper (transactional) storage and efficient data retrieval support.
- *Data ingestion* enables importing and exporting the data over the respective system.
- *Big Data processing* supports the efficient and clustered processing of Big Data by executing the main logic of the user application(s).
- *Resource management* is responsible for the proper and efficient management of computational resources.

In this article, our focus is mainly on the *Data storage* and *Data placement* parts of the above architecture. Our rationale is that the integration of such parts (or Big Data lifecycle management phases) covers the core of a DLMS. An application's data access workflow in the Cloud is presented in Fig. 2. As a first step, the application checks the availability of the input data. In general, the data needs to be known by the system to optimally handle it. It maps to two main cases: (i) data already exist and have been registered; (ii) data do not exist and must be registered. In the latter case, metadata is needed

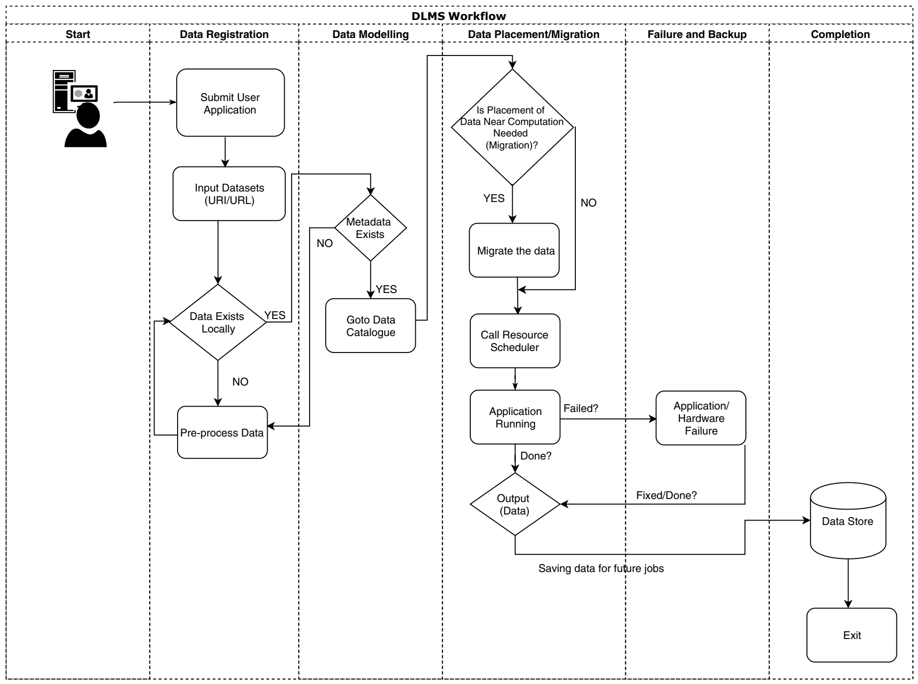

Fig. 2 Standard workflow of application data lifecycle

to register the data into the system (thus mapping to the data-registration process). During the data modelling (see "Data modelling" sub-section), the metadata are maintained via a data catalogue (i.e., a special realisation of *Metadata management* component). Such an approach can guarantee the efficient maintenance of application data throughout the application's lifecycle by both knowing and dynamically altering the values of data features (such as data type, size, location, data format, user preference, data replica numbers, cost constraints) whenever needed. In the next phase, based on the employed data placement methodology, the data is placed/migrated next to the application or both the data and application code is collocated. Here, the underlying scheduler (realising the *Data placement* component) acquires the up-to-date data knowledge to achieve an efficient data placement during both the initial application deployment and its runtime. Such an approach can restrain unnecessary data movement and reduces cost (at runtime) [30–32]. Next, during the application execution, two situations may arise: (i) new data sets are generated; (ii) data sets are transformed into another form (such as data compression). Furthermore, temporary data may also need to be handled. Finally, once application execution ends, the generated or transformed data needs to be stored (or backed up) as per user instructions.

In general, a hierarchical storage management [33] could be considered as a DLMS tool. In recent times, cognitive data management (CDM) has gained industrial support for automated data management together with high-grade efficiency. The CDM (e.g., Stronglink[1]) is generally the amalgamation of intelligent (artificial-intelligence[2]/machine learning-based approach) distributed storage including resource management together with a more sophisticated DLMS component. The CDM works on the database-as-a-service (DBaaS) layer which instructs the data to be used by the scheduler with an efficient management approach including the exploitation of the data catalogue via data modelling.

Methodology

We have conducted a systematic literature review (SLR) on Big Data placement and storage methods in the Cloud, following the guidelines proposed in [34]. Such an SLR comprises three main phases: (i) SLR planning, (ii) SLR conduction and (iii) SLR reporting. In this section, we briefly discuss the first two phases. While the remaining part of this manuscript focuses on the presenting the survey, the identification of the remaining research issues and the potential challenges for current and future work.

SLR planning

This phase comprises three main steps: (i) SLR need identification, (ii) research questions identification and (iii) SLR protocol formation.

SLR need identification

Here, we are advocating to add more focus on the Big Data storage and placement phases of the respective Big Data management lifecycle. Thus be able to confront the respective challenges that Big Data place on them. Such phases are also the most crucial in the attempt to satisfy the non-functional requirements of Big Data applications. The primary focus of this survey is over storage and placement phases. It is an attempt to examine if they are efficiently and effectively realised by current solutions and approaches. The twofold advantage of identifying the efficient ways to manage and store Big Data are: (i) practitioners can select the most suitable Big Data management solutions for satisfying both their functional and non-functional needs; (ii) researchers can fully comprehend the research area and identify the most interesting directions to follow. To this end, we are countering both the data placement and the storage issues focusing on the Big Data management lifecycle and Cloud computing under the prism of non-functional aspects. In contrast to previous surveys that have concentrated mainly on the Big Data storage issues in the context of functional aspects.

Research questions identification

This survey has the ambition to supply suitable and convincing answers to:

1. What are the most suitable (big) data storage technologies and how do they compete with each other according to certain criteria related to non-functional aspects?
2. What are the most suitable and sophisticated (big) data placement methods that can be followed to (optimally) place and/or migrate Big Data?

[1] https://strongboxdata.com/products/stronglink/.

[2] https://www.ibm.com/services/artificial-intelligence.

Table 1 Search query

(Big Data) AND (METHODOLOGY OR METHOD OR ALGORITHM OR APPROACH OR SURVEY OR STUDY) AND (MANAGEMENT OR PLACEMENT OR POSITION OR ALLOCATION OR STORAGE) WITH TIME SPAN:2010–2018

SLR protocol formation

It is a composite step related to the identification of (i) (data) sources—here we have primarily consulted the Web of Science and Scopus, and (ii) the actual terms for querying these (data) sources—here, we focus on population, intervention and outcome as mentioned in [34]. It is worth to note that such data sources supply nice structured searching capabilities which enabled us to better pose the respective query terms. The population mainly concerns target user groups in the research area or certain application domains. The intervention means the specific method employed to address a certain issue (used terms include: methodology, method, algorithm, approach, survey and study). Lastly, the outcome relates to the final result of the application of the respective approach (such as management, placement, positioning, allocation, storage). Based on these terms, the abstract query concretised in the context of the two data sources can be seen in Table 1.

SLR conduction

Systematic literature review conduction includes the following steps: (i) study selection criteria; (ii) quality assessment criteria; (iii) study selection procedure. All these steps are analysed in the following three paragraphs.

Study selection

The study selection was performed via a certain set of inclusion and exclusion criteria. The inclusion criteria included the following:

- Peer-reviewed articles.
- Latest articles only (last 8 years).
- In case of equivalent studies, only the one published in the highest rated journal or conference is selected to sustain only a high-quality set of articles on which the review is conducted.
- Articles which supply methodologies, methods or approaches for Big Data management.
- Articles which study or propose Big Data storage management systems or databases.
- Articles which propose Big Data placement methodologies or algorithms.

While the exclusion criteria were the following:

- Inaccessible articles.
- Articles in a different language than English.
- Short papers, posters or other kinds of small in contribution articles.
- Articles which deal with the management of data in general and do not focus on Big Data.

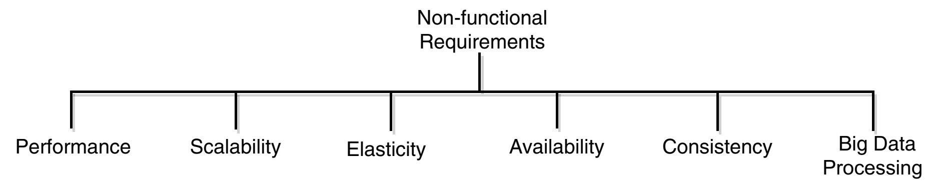

Fig. 3 Non-functional features of data management systems

- Articles that focus on studying or proposing normal database management systems.
- Articles that focus on studying or proposing normal file management systems.
- Articles that focus on the supply of Big Data processing techniques or algorithms. As the focus in this article is mainly on how to manage the data and not how to process them to achieve a certain result.

Quality assessment criteria

Apart from the above criteria, quality assessment criteria were also employed to enable prioritising the review as well as possibly excluding some articles not reaching certain quality standards. In the context of this work, the following criteria were considered:

- Presentation of the article is clear and there is no great effort needed to comprehend it.
- Any kind of validation is offered especially in the context of the proposal of certain algorithms, methods, systems or databases.
- The advancement over the state-of-the-art is clarified as well as the main limitations of the proposed work.
- The objectives of the study are well covered by the approach that is being employed.

Study selection procedure

It has been decided to employ two surveyors for each main article topic which were given a different portion of the respective reviewing work depending on their expertise. In each topic, the selection results of one author were assessed by the other one. In case of disagreement, a respective discussion was conducted. If this discussion was not having a positive outcome, the respective decision was delegated to the principal author which has been unanimously selected by all authors from the very beginning.

Non-functional data management features

For effective Big Data management, current data management systems (DMSs), including distributed file systems (DFSs) and distributed database management systems (DDBMSs) need to provide a set of non-functional features to cater the storage, management and access of the continuously growing data. This section introduces a classification of the non-functional features (see Fig. 3) of DMSs in the Big Data domain extracted from [10, 13, 35–37].

Figure 3 provides an overview of the relevant non-functional features while the following subsections attempt to analyse each of them.

Performance

Performance is typically referred to as one of the most important non-functional features. It directly relates to the execution of requests by the DMSs [38, 39]. Typical performance metrics are *throughput* and *latency*.

Scalability

Scalability focuses on the general ability to process arbitrary workloads. A definition of scalability for distributed systems in general and with respect to DDBMSs is provided by Agrawal et al. [40], where the terms scale-up, scale-down, scale-out and scale-in are defined focusing on the management of growing workloads. Vertical as well as horizontal scaling techniques are applied to distributed DBMSs and can also be applied to DFSs. Vertical scaling applies by adding more computing resources to a single node. While horizontal scaling applies by adding nodes to a cluster (or in general to the instances of a certain application component).

Elasticity

Elasticity is tightly coupled to the horizontal scaling and helps to overcome the sudden workload fluctuations by scaling the respective cluster without any downtime. Agrawal et al. [40] formally define it by focusing on DDBMSs as follows *"Elasticity, i.e. the ability to deal with load variations by adding more resources during high load or consolidating the tenants to fewer nodes when the load decreases, all in a live system without service disruption, is therefore critical for these systems"*. While elasticity has become a common feature for DDBMSs, it is still in an early stage for DFSs [41].

Availability

The availability tier builds upon the scalability and elasticity as these tiers are exploited to handle request fluctuations [42]. Availability represents the degree to which a system is operational and accessible when required. The availability of a DMS can be affected by *overloading at the DMS layer* and/or *failures at the resource layer*. During overloading, a high number of concurrent client requests overload the system such that these requests are either handled with a non-acceptable latency or not handled at all. On the other hand, a node can fail due to a resource failure (such as network outage or disk failure). An intuitive way to deal with overload is to scale-out the system. Distributed DMSs apply data replication to handle such resource failures.

Consistency

To support high availability (HA), consistency becomes an even more important and challenging non-functional feature. However, there is a trade-off among consistency, availability and partitioning guarantees, inscribed by the well-known CAP theorem [43]. This means that different kinds of consistency guarantees could be offered by a DMS. According to [44] consistency can be considered from both the client and data perspectives (i.e., from the DMS administrator perspective). The client-centric consistency can be classified further into staleness and ordering [44]. Staleness defines the lagging of replica behind its master. It can be measured either in time or versions. Ordering defines that all requests must be executed on all replicas in the same chronological order. Data-centric consistency focuses on the synchronization processes among replicas and the internal ordering of operations.

Big Data processing

The need of native integration of (big) data processing frameworks into the DMSs arises along with the number of recently advanced Big Data processing frameworks, such as Hadoop MapReduce, Apache Spark, and their specific internal data models. Hence, the DMSs need to provide native drivers for Big Data processing frameworks which can automate the transformation of DMS data models into the respective Big Data processing framework storage models. Further, these native drivers can exploit data locality features of the DMSs as well. Please note that such a feature is also needed based on the respective DLMS architecture that has been presented in "Data lifecycle management (DLM)" section as a Big Data processing framework needs to be placed on top of the data management component.

Data storage systems

A DLMS in the Big Data domain requires both the storage and the management of heterogeneous data structures. Consequently, a sophisticated DLMS would need to support a diverse set of DMSs. DMSs can be classified into file systems for storing unstructured data and DBMSs (database management systems) for storing semi-structured and structured data. However, the variety of semi-structured and structured data requires suitable data models (see Fig. 4) to increase the flexibility of DBMSs. Following these requirements, the DBMS landscape is constantly evolving and becomes more heterogeneous.[3] The following sub-sections provides (i) an overview of related work on DBMS classifications; (ii) a holistic and up-to-date classification of current DBMS data models; (iii) a qualitative analysis of selected DBMSs; (iv) a classification and analysis of relevant DFSs.

Database management systems

The classification of the different data models (see Fig. 4) for semi-structured data has been in the focus since the last decade [37] as heterogeneous systems (such as Dynamo, Cassandra [45] and BigTable [46]) appeared on the DBMS landscape. Consequently, the term NoSQL evolved, which summarizes the heterogeneous data models for semi-structured data. Similar, the structured data model evolved with the NewSQL DBMSs [13, 47].

Several surveys have reviewed NoSQL and NewSQL data models over the last years and analyze the existing DBMS with respect to their data models and the specific non-functional features [11, 13, 35–37, 48, 49]. In addition, dedicated surveys focus explicitly specific data models (such as the time series data model [50, 51]) or specific DBMS architectures (such as in-memory DBMS [14]).

[3] http://nosql-database.org/ lists over 225 DBMS for semi-structured data.

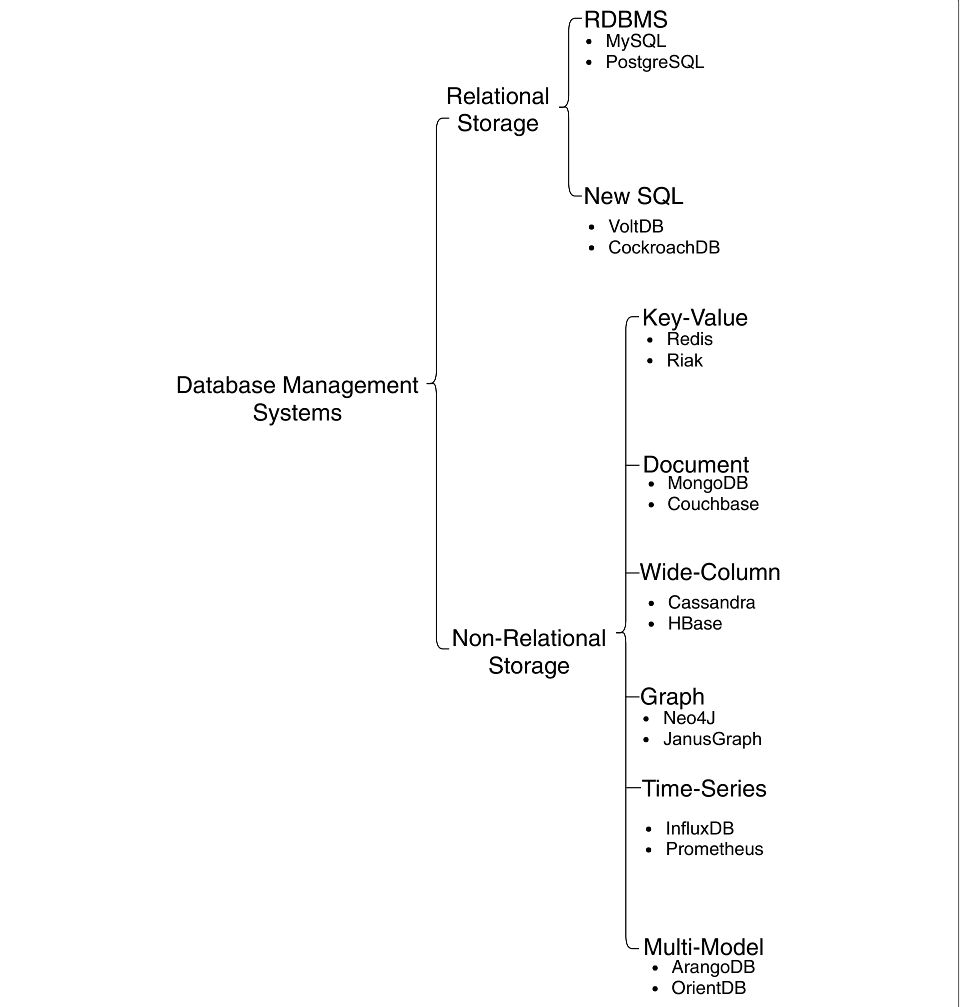

Fig. 4 DBMS data model classification

Cloud-centric challenges for operating distributed DBMS are analysed by [13], considers the following: horizontal scaling, handling elastic workload patterns and fault tolerance. It also classifies nineteen DDBMSs against features, such as partitioning, replication, consistency and security.

Recent surveys on NoSQL-based systems [35, 49] derive both, the functional and the non-functional NoSQL and NewSQL features and correlated them with distribution mechanisms (such as sharding, replication, storage management and query processing). However, the implications of Cloud resources or the challenges of Big Data applications were not considered. Another conceptual analysis of NoSQL DBMS is carried out by [48]. It outlines many storage models (such as key-value, document, column-oriented and graph-based) and also analyses current NoSQL implementations against persistence, replication, sharding, consistency and query capability. However, recent DDBMSs (such as time-series DBMSs or NewSQL DBMSs) are not analysed from Big Data as well as the Cloud context. A survey on DBMS support for Big Data with the focus on data storage models, architectures and consistency models is presented by [11]. Here, the relevant DBMSs are analysed towards their suitability for Big Data applications, but the Cloud service models and evolving DBMSs (such as time-series databases) are also not considered.

An analysis of the challenges and opportunities for DBMSs in the Cloud is presented by [52]. Here, the relaxed consistency guarantees (for DDBMS) and heterogeneity, as well as the different level of Cloud resource failures are explained. Moreover, it is also explicated that HA mechanism is needed to overcome failures. However, the HA and horizontal scalability come with the weaker consistency model (e.g., BASE [53]) compared to ACID [43].

In the following, we distil and join existing data model classifications (refer to Fig. 4) into an up-to-date classification of the still-evolving DBMS landscape. Hereby, we select relevant details for the DLMS of Big Data applications, while we refer the interested reader to the presented surveys for an in-depth analysis of specific data models. Analogously, we apply a qualitative analysis of currently relevant DBMS based on the general DLMS features (see "Non-functional data management features" section), while in-depth analysis of specific features can be found in the presented surveys. Hereby, we select two common DBMS[4] of each data model for our analysis.

Relational data models

The relational data model stores data as tuples forming an ordered set of attributes; which can be extended to extract more meaningful information [54]. A relation forms a table and tables are defined using a static, normalised data schema. SQL is a generic data definition, manipulation and query language for relational data. Popular representative DBMSs with a relational data model are MySQL and PostgreSQL.

NewSQL

The traditional relational data model provides limited data partitioning, horizontal scalability and elasticity support. NewSQL DBMSs [55] aim at bridging this gap and build upon the relational data model and SQL. However, NewSQL relaxes relational features to enable horizontal scalability and elasticity [13]. It is worth to note that only a few NewSQL DBMSs, such as VoltDB[5] and CockroachDB,[6] are built upon such architectures with the focus on scalability and elasticity as most NewSQL DBMSs are constructed out of existing DBMSs [47].

Key-value

The key-value data model relates to the hash tables of programming languages. The data records are tuples consisting of key-value pairs. While the key uniquely identifies an entry, the value is an arbitrary chunk of data. Operations are usually limited to simple put or get operations. Popular key-value DBMSs are Riak[7] and Redis.[8]

[4] https://db-engines.com/en/ranking.

[5] https://www.voltdb.com/.

[6] https://www.cockroachlabs.com/.

[7] http://basho.com/products/riak-kv/.

[8] https://redis.io/.

Document

The document data model is similar to the key-value data model. However, it defines a structure on the values in certain formats, such as XML or JSON. These values are referred to as documents, but usually without fixed schema definitions. Compared to key-value stores, the document data model allows for more complex queries as document properties can be used for indexing and querying. MongoDB[9] and Couchbase[10] represent the common DBMSs with a document data model.

Wide-column

The column-oriented data model stores data by columns rather than by rows. It enables both storing large amounts of data in bulk and efficiently querying over very large structured data sets. A column-oriented data model does not rely on a fixed schema. It provides nestable, map-like structures for data items which improve flexibility over fixed schema [46]. The common representatives of column-oriented DBMSs are Apache Cassandra[11] and Apache HBase.[12]

Graph

The graph data model primarily uses graph structures, usually including elements like nodes and edges, for data modelling. Nodes are often used for the main data entities, while edges between nodes are used to describe relationships between entities. Querying is typically executed by traversing the graph. Typical graph-based DBMS are Neo4J[13] and JanusGraph.[14]

Time-series

The time-series data model [50] is driven by the needs of sensor storage within the Cloud and Big Data context. The time-series DBMSs are typically built upon existing non-relational data models (preferably key-value or column-oriented), and add a dedicated time-series data model on top. The data model is built upon data points which comprise a time stamp, an associated numeric value and a customisable set of metadata. Time-series DBMSs offers analytical query capabilities, which cover statistical functions and aggregations. Well-known time-series DBMSs are InfluxDB[15] and Prometheus.[16]

Multi-model

A multi-model address the problem of polyglot persistence [56] which signifies that each of the existing non-relational data models addresses a specific use case. Hence, multi-model DBMSs combine different data models into a single DBMS while build upon one storage backend to improve flexibility (e.g., providing the document and graph data

[9] https://www.mongodb.com/.

[10] https://www.couchbase.com/.

[11] http://cassandra.apache.org/.

[12] https://hbase.apache.org/.

[13] https://neo4j.com/.

[14] http://janusgraph.org/.

[15] https://www.influxdata.com/

[16] https://prometheus.io/.

model via a unified query interface). Common multi-model DBMSs are ArangoDB[17] and OrientDB.[18]

Comparison of selected DBMSs

In this section, we analyse already mentioned DBMSs in the context of Big Data applications (see Table 2). To perform this, we first analyse already mentioned DBMS (of the previously introduced data models) with respect to their features and supported Cloud service models. Next, we provide a qualitative analysis with respect to the non-functional features of the DMSs (refer to "Non-functional data management features" section). For quantitative analysis of these non-functional requirements, we refer the interested reader to the existing work focused on DBMS evaluation frameworks [44, 57–60] and evaluation results [42, 61, 62].

Qualitative criteria

In the Table 2, the first three columns present each DBMS and its data model, followed by the technical features and the service models supported. The analysis only considers the standard version of a DBMS.

In the following, we attempt to explicate each of the technical features considered. The DBMS *architecture* is classified into single, master–slave and multi-master architectures [56]. The *sharding* strategies are analysed based on the DBMS architectures; they can be supported manually as well as automatically in a hash- or range-based manner. The *elasticity* feature relies on a distributed architecture and relates to whether a DBMS supports adding and/or removing nodes from the cluster at runtime without a downtime. For consistency and availability guarantees, each DBMS is analysed with respect to its consistency (C), availability (A) and partition tolerance (P) properties within the *CAP* theorem (i.e., CA, CP, AC or AP) [43]. However, it should be highlighted that we did not consider fine-grained configuration options that might be offered for a DBMS to vary the CAP properties. Next, the *replication* mechanisms are analysed in terms of both cluster and cross-cluster replication (also known as geo-distribution). Consequently, a DBMS supporting cross-cluster replication implicitly supports cluster replication. The interested reader might consider [63] for more fine-grained analysis of replication mechanisms of DDBMSs. The *Big Data adapter* is analysed by evaluating native and/or third-party drivers for Big Data processing frameworks. Finally, the DDBMSs are classified based on their offering as *community* editions, *enterprise* commercial editions or managed *DBaaS*. One exemplary provider is presented if the DBMS is offered as a DBaaS.

Qualitative analysis

The resulting Table 2 represents the evolving landscape of the DBMSs. The implemented features of existing DBMSs significantly differ (except the RDBMSs) even within one data model. The heterogeneity of analysed DBMSs is even more obvious across data models. Further, the heterogeneous DBMS landscape offers a variety of potential DBMS solutions for Big Data.

[17] https://www.arangodb.com/.

[18] https://orientdb.com/.

Table 2 Technical feature analysis of selected DBMS

DBMS	Version	Data model	Technical features						Service model		
			Architecture	Sharding	Elasticity	CAP	Replication	Big Data adapter	Community	Enterprise	DBaaS
MySQL	8.0.11	RDBMS	Single/master–slave	Manual	No	CA	Cluster	3rd party (SQL-based)	Yes	Yes	https://Cloud.oracle.com/ mysql
PostgreSQL	10.4	RDBMS	Single/master–slave	Manual	No	CA	Cluster	3rd party (SQL-based)	Yes	Yes	https://aws.amazon.com/rds/postgresql/
VoltDB	8.1.2	NewSQL	Multi-master	Hash	Yes (commercial)	CP	Cross-cluster (commercial)	3rd party (SQL-based)	Yes	Yes	No
CockroachDB	2.0.3	NewSQL	Multi-master	Hash	Yes	CP	Cross-cluster (commercial)	3rd party (SQL-based)	Yes	Yes	No
Riak	2.2.3	Key-value	Multi-master	Hash	Yes	AP	Cross-cluster	Native	Yes	Yes	No
Redis	4.0	Key-value	Multi-master	Hash	Yes	AC	Cluster	Native	Yes	Yes	https://redislabs.com/
MongoDB	4.0.0	Document	Multi-master	Hash/range	Yes	CP	Cross-cluster	Native	Yes	Yes	https://www.mongodb.com/ Cloud/atlas
Couchbase	5.0.1	Document	Multi-master	Hash	Yes	CP	Cross-cluster	Native	Yes	Yes	https://www.couchbase.com/products/Cloud/managed-Cloud
Cassandra	3.11.2	Wide-column	Multi-master	Hash/range	Yes	AP	Cross-cluster	Native	Yes	Yes, By DataStax	https://www.instaclustr.com/solutions/managed-apache-cassandra/
HBase	2.0.1	Wide-column	Multi-master	Hash	Yes	CP	Cross-cluster	3rd party	Yes	Yes, By Cloudera	No
Neo4J	3.4.1	Graph	Master–slave	No	Yes	CA	Cross-cluster	Native	Yes	Yes	https://www.graphstory.com/
JanusGraph	0.2.0	Graph	Multi-master	Manual	Yes	AP/CP	Cluster	3rd party	Yes	No	No

Table 2 (continued)

DBMS	Version	Data model	Technical features						Service model		
			Architecture	**Sharding**	**Elasticity**	**CAP**	**Replication**	**Big Data adapter**	**Community**	**Enterprise**	**DBaaS**
ArangoDB	3.3.11	Multi-model (key-value, document, graph)	Multi-master	Hash	Yes	CP	Cross-cluster	Native	Yes	Yes	No
OrientDB	3.0.2	Multi-model (key-value, document, graph)	Multi-master	Hash	Yes	–	Cross-cluster (commercial)	Native	Yes	Yes	No
InfluxDB	1.5.4	Time-series	Multi-master (commercial)	Range	Yes (commercial)	AP/CP	Cross-cluster (commercial)	3rd party	Yes	Yes	https://cloud.influxdata.com/
Prometheus	2.3	Time-series	Master–slave	Manual	No	–	Cluster	3rd party	Yes	Yes	No

The feature analysis provides a baseline for the qualitative analysis of the non-functional features. From the (horizontal) scalability point-of-view, a DBMS with a multi-master architecture is supposed to provide scalability for write and read workloads, while a master–slave architecture is supposed to provide read scalability. Due to the differences between the DBMSs, the impact of elasticity requires additional qualitative evaluations [42].

The consistency guarantees correlate to the classification in the CAP theorem. Table 2 clearly shows the heterogeneity compared to the consistency guarantees. Generally, the single-master or master–slave architectures provide strong consistency guarantees. Multi-master architectures cannot be exactly classified into the CAP theorem as their consistency guarantees heavily depend on the DBMS runtime configuration [64]. Additional evaluations of the consistency for the selected DBMSs are required for strong consistency (so as to ensure scalability, elasticity and availability) [44, 62].

Providing HA directly relates to the supported replication mechanisms to overcome failures. The analysis shows that all DBMSs support cluster-level replication, while cross-cluster replication is supported by ten out of the sixteen DBMSs. Big Data processing relates to the technical feature of Big Data adapters. Table 2 clearly shows that seven DBMSs provide native adapters and nine DBMS enable it via third-party adapters to support Big Data processing. The service model of all the DBMSs is either available as a self-hosted community or enterprise version. In addition, both RDBMS and six NoSQL DBMS are offered as managed DBaaS. While the DBaaS offerings are abstracting all operational aspects of the DBMS, an additional analysis might be required with respect to their non-functional features and cost models [65].

Cloudification of DMS

Traditional on-premise DBMS offerings are still popular, but the current trend shows that DDBMSs running in the Cloud are also well-accepted. Especially, as Big Data imposes new challenges such as scalability, the diversity of data management or the usage of Cloud resources, towards the massive storage of data [66]. In general, the distributed architecture of DMSs evolved their focus over exploiting the Cloud features and catering the 5Vs of Big Data [67]. Data-as-a-service (DaaS) mostly handles the data aggregation and management via appropriate web-services, such as RESTful APIs. While database-as-a-service (DBaaS) offers database as a service which can include (distributed) a relational database or a non-relational one. In most of the cases, storage-as-a-service (STaaS) includes both DaaS and the DBaaS [68]. Furthermore, BDaaS [3] is a Cloud service (such as Hadoop-as-a-service) where traditional applications are migrated from local installations to the Cloud. BDaaS wraps three primary services. They are (i) IaaS (for underlying resources), (ii) STaaS (a sub-domain of platform-as-s-service (PaaS)) for managing the data via dynamic scaling and (iii) data management (such as data placement, replica management).

Distributed file systems

A distributed file systems (DFS) is an extended networked file system that allows multiple distributed nodes to internally share data/files without using remote call methods or procedures [69]. A DFS offers scalability, fault-tolerance, concurrent file access and

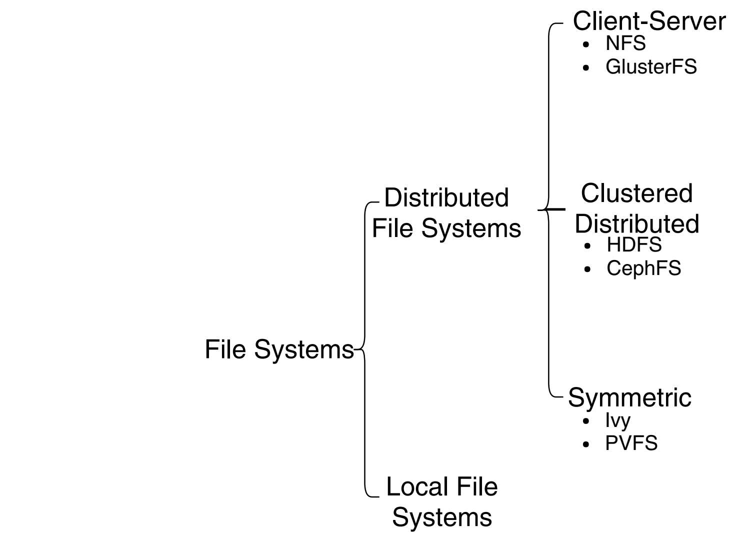

Fig. 5 Distributed file systems classification

metadata support. However, the design challenges (independent of data size and storage type) of a DFS are transparency, reliability, performance, scalability, and security. In general, DFSs do not share storage access at the block level but rather work at the network level. In DFSs, security relies on either access control lists (ACLs) or respectively defined capabilities, depending on how the network is designed. DFSs can be broadly classified into three models and respective groups (see Fig. 5). First, client–server architecture based file systems which supply a standardized view of a local file system. Second, clustered-distributed file systems which offer multiple nodes to enable concurrent access to the same block device. Third, symmetric file systems, where all nodes have a complete view of the disk structure. Below, we briefly analyse each category in a separate sub-section while we also supply some strictly open-source members for it.

Client–server model

In the client–server architecture based file system, all communications between servers and clients are conducted via remote procedure calls. The clients maintain the status of current operations on a remote file system. Each file server provides a standardized view of its local file system. Here, the file read-operations are not mutually exclusive but the write operations are. File sharing is based on mounting operations. Only the servers can mount directories exported from other servers. Network File System (NFS) and GlusterFS[19] are two popular open source implementations of the client–server model.

Clustered-distributed model

Clustered-distributed based systems organize the clusters in an application-specific manner and are ideal for DCs. The model supports a huge amount of data; the data is stored/partitioned across several servers for parallel access. By design, this DFS model

[19] https://docs.gluster.org/en/latest/.

Table 3 Feature analysis of selected DFSs

DFS	Version	FileSystem	Technical features					
			Architecture	Sharding	Elasticity	CAP	Replication	Big Data adapter
NFS	4.2	Client–server	Fully-centralized	Index/range	No	CA	Block level	3rd party
GlusterFS	4.0	Client–server	Fully-centralized	Automatic	Yes	CA	Node level	Native
HDFS	3.0.1	Clustered-distributed	Less-centralized	Fixed size	Yes	AP	Block level	Native
CephFS	12.2.5	Clustered-distributed	Less-centralized	Index/range	Yes	CP	Cluster-level	Native/3rd party
Ivy	0.3	Symmetric	Fully-distributed	DHash	Yes	AP	Block-level	–
PVFS	2.0	Symmetric	Fully-distributed	Hash	Yes	AP	–	3rd party

is fault tolerant as it enables the hosting of a number of replicas. Due to the huge volume of data, data are appended instead of overwritten. In general, the DNS servers map (commonly using round-robin fashion) access requests to the clusters for load-balancing purposes. The Hadoop distributed file system (HDFS) and CephFS[20] are two popular implementations of such a DFS model.

Symmetric model

Symmetric is a DFS that supports a masterless architecture, where each node has the same set of roles. It mainly resembles a peer-to-peer system. In general, the symmetric model employs a distributed hash table approach for data distribution and replication across systems. Such a model offers higher availability but reduced performance. Ivy [70] and the parallel virtual file system (PVFS) [71] are examples of a symmetric DFS model.

DFS evaluation

Similar to DDBMSs, we also compare the open source implementations of DFSs according to the same set of technical features or criteria. A summary of this comparison is depicted in Table 3. In general, most of the file systems are distributed in nature (except NFS and GlusterFS). However, they do exhibit some architectural differences. NFS and GlusterFS are both developed focusing on a master–slave approach, while Ivy and PVFS are based on the masterless model. Data partitioning (or sharding) is also supported dynamically (featured by Ivy and PVFS) or statically via a fixed size (as in case of HDFS) by these DFSs. Elasticity or supporting the data scaling is a very important feature for many Big Data applications (especially hosted at Cloud). We can thus observe that except NFS all mentioned DFSs support scalability. Further, HDFS, CephFS, Ivy and PVFS are fault tolerant as well. Replication, highly needed for not losing data, is well supported by all DFSs. However, their granularity differs from the block to the cluster level. Finally, these DFSs also offer some form of hooks (either native or third-party supplied) to be used with Big Data frameworks.

[20] http://docs.ceph.com/docs/mimic/cephfs/.

Data placement techniques

In the Cloud ecosystem, traditional placement algorithms incur a high cost (including the time) on storing and transferring data [72]. Placing data while data is partitioned and distributed across multiple locations is a challenge [23, 73]. Runtime data migration is an expensive affair [30–32] and the complexity increases due to the frequent change of applications as well as DCs' behaviour (i.e., resources or latencies) [74]. Placing a large amount of data across the Cloud is complex due to issues, such as (i) data storage and transfer cost optimisation while maintaining data dependencies; (ii) data availability and replication; (iii) privacy policies, such as restricted data storage based on geo-locations. Data replication can influence consistency, while it also enhances the scalability and higher availability of data. In general, the existing data placement strategies can be grouped based on user-imposed constraints, such as data access latency [75], fault tolerance [76], energy-cost awareness [77], data dependency [78, 79] and robustness or reliability [80, 81].

Formal definition

The data placement in a distributed computing domain is an instance of NP-hard problem [82], while it can be reduced to a bin-packing problem instance. Informally, the data placement problem can be described as follows: given a certain workflow, the current data placement, and a particular infrastructure, find the right position(s) of data within the infrastructure to optimise one or more certain criteria, such as the cost of the data transfer.

A formal representation of this problem as follows: suppose that there are N datasets, represented as d_i (where $i = 1, \ldots, N$). Each dataset has a certain size s_i. Further, suppose that there are M computational elements represented as V_j (where $j = 1, \ldots, M$). Each computational element has a certain storage capacity denoted as c_j. Finally, suppose that there is a workflow W with T tasks which are represented as t_k (where $k = 1, \ldots, T$). Each task has a certain input $t_k.input$ and output $t_k.output$, where each maps to a set of datasets.

The main set of decision variables is c_{ij} representing the decisions (e.g., based on privacy or legal issues) of whether a certain dataset i should be stored in a certain computational element j. Thus, there is a need to have $c_{ij} == 1$ for each i and a certain j. Two hard constraints need to hold: (i) a dataset should be stored in one computational element which can be represented as follows: $\sum_j c_{ij} = 1$ for each i. It is worth to note that this constraint holds when no dataset replication is allowed. Otherwise, it would take the following form: $\sum_j c_{ij} >= r$, where r is the replication factor; (ii) the capacity of a computational element should be sufficient for hosting the respective dataset(s) assigned to it. This is represented as follows: $\sum_i c_{ij} * s_i <= c_j$ for each j.

Finally, suppose that the primary aim is to reduce the total amount of data transfers for the whole workflow. In this respect, this optimisation objective can be expressed as follows:

$$\text{minimise} \sum_k \sum_{d_i, d_{i'} \in t_k.input} \left(\mathrm{m}(d_i) <> \mathrm{m}\left(d_{i'}\right)\right) \tag{1}$$

where $\mathrm{m}(d_i)$ (which supplies as the output a value in $[1, M]$) indicates the index of the computational element that has been selected for a certain dataset. This objective adds the amount of data transfers per each workflow task which relates to the fact that the task will be certainly placed in a specific resource mapping to one of the required input datasets. Thus, during its execution, the data mapping to the rest of the input datasets will need to be moved in order to support the respective computation needed.

Data placement methodologies

We broadly classify the proposed data placement methods into data dependency, holistic task and data scheduling and graph-based methods. The methods in each category are analysed in the following subsections.

Data dependency methods

A data-group-aware placement scheme is proposed in [83] by employing the bond energy algorithm (BEA) [84] to transform the original data dependency matrix into a Hadoop cluster. It exploits access patterns to find an optimal data grouping to achieve better parallelism and workload balancing. In [85], a data placement algorithm is proposed for solving the data inter-dependency issue at the VM level. Scalia [86] proposes a Cloud storage brokerage scheme that optimises the storage cost by exploiting the real-time data access patterns. Zhao et al. [87] proposed data placement strategies for both initial data placement and relocation using a particle swarm optimization (PSO) algorithm. For fixed data set placement, this method relies on hierarchical data correlation and performs data re-allocation during the storage saturation. Yuan et al. [78] propose a k-means based dataset clustering algorithm to construct a data dependency matrix by exploiting the data dependency and the locality of computation. Later, the dependency matrix is transformed by applying the BEA while items are clustered based on their dependencies by following a recursive binary partitioning algorithm. In general, the preservation of time locality can significantly impact caching performance while the efficient re-ordering of jobs can improve the resource usage. In [79] authors propose a file grouping policy for pre-staging data by preserving time locality and enforcing the role of job re-ordering via extracting access patterns.

Task and data scheduling methods

In [88], the authors propose an adaptive (based on multi-objective optimization model) data management middleware which collects system-state information and abstracts away the complexities of multiple Cloud storage systems. For internet-of-things (IoT) data streaming support, Lan et al. [89] proposed a data stream partitioning mechanism by exploiting statistical feature extraction. Zhang et al. [90] propose a mixed-integer linear programming model for modelling the data placement problem. It considers both the data access cost as well as the storage limitations of DCs. Hsu et al. [91] proposed a Hadoop extension by adding dynamic data re-distribution (by VM profiling) before the map phase and VM mapping for reducers based on partition size and VM availability. Here, high capacity VMs are assigned for high workload reducers. Xu et al. [92] proposes a genetic programming approach to optimise the overall number of data transfers. However, this approach does not consider the DCs' capacity constraints and the non-replication constraints of data sets. In [93], a policy engine is proposed for managing both the number of parallel streams (between origin and destination nodes) and the priorities for data staging jobs in scientific workflows. The policy engine also considers data transfers, storage allocation and network resources.

The storage resource broker [23] provides seamless access to the different distributed data sources (interfacing multiple storages) via its APIs. It works as a middleware between the multiple distributed data storages and applications. BitDew [94] offers a programmable environment for data management via metadata exploitation. The data scheduling (DS) service takes care of implicit data movement. Pegasus [95] provides a framework that maps complex scientific applications onto distributed resources. It stores the newly generated data and also registers them in the metadata catalogue. The replica location service [96] is a distributed, scalable, data management service that maps the logical data names to target names. It supports both centralized as well as distributed resource mapping. Kosar and Livny [81] proposes a data placement that consists of a scheduler, a planner and a resource broker. The resource broker is responsible for matching resources, data identification and decisions related to data movement. The scheduling of data placement jobs relies on the information given by the workflow manager, the resource broker and the data miner. A very interesting feature of the proposed sub-system is that it is able to support failure recovery through the application of retry semantics.

Graph-based data placement

Yu and Pan [72] proposes the use of *sketches* to construct a hyper-graph sparsifier of data traffic to lower the data placement cost. Such sketches represent data structures that approximate properties of a data stream. LeBeane et al. [97] proposed on-line graph-partitioning multiple strategies to optimise data-ingress across heterogeneous clusters. SWORD [98] handles the partitioning and placement for OLTP workloads. Here, the workload is represented as a hypergraph and a hyper-graph compression technique is employed to reduce the data partitioning overhead. An incremental data re-partitioning technique is also proposed that modifies data placement in multiple steps to support workload changes. Kayyoor et al. [99] propose how to map nodes to a subset of clusters via satisfying user constraints. It minimises the query span for query workloads by applying replica selection and data placement algorithms. The query-based workload is represented as hyper-graphs and a hypergraph partitioning algorithm is used to process them. Kaya et al. [100] model the workflow as a hypergraph and employ a partitioning algorithm to reduce the computational and storage load while trying to minimise the total amount of file transfers.

Comparative evaluation

In this section, we have carefully selected a set of criteria to evaluate the methods analysed in "Data placement methodologies" section. The curated criteria are: (i) *fixed data sets*—whether the placement of data can be a priori fixed in sight of, e.g., regulations, (ii) *constraint satisfaction*—which constraint solving technique is used, (iii) *granularity*—what is the granularity of the resources considered, (iv) *intermediate data handling*—whether intermediate data, produced by, e.g., a running workflow, can be also handled, (v) *multiple application handling*—whether the data placement over multiple applications can be supported, (vi) *increasing data size*—whether the growth rate of data is taken into account, (vii) *replication*—whether data replication is supported, (viii) *optimisation criteria*—which optimisation criteria are exploited, (ix) *additional system related*

Table 4 Comparative summary of existing data placement algorithms

Approach	Fixed DS	Constraint satisfaction	Granul.	Interm. DS	Mult. appl.	Data size	Repl.	Opt. criteria	Add. info.
BDAP [85]	Yes	Meta-heuristic	Fine	Yes	No	No	No	Comm. cost	No
Xu [92]	No	Meta-heuristic	Coarse	No	No	No	No	Data transf. number	No
Yuan [78]	Yes	Recursive binary part.	Coarse	Yes	Yes	Yes	No	Data transf. number	No
Kaya [100]	No	Hypergraph part.	Coarse	No	No	No	No	Data transf. number	No
Zhao [87]	Yes	Hierarchical part. clust. + PSO	Fine	Yes	No	No	No	Data transf. number	No
Wang [83]	No	Recursive clust. + ODPA	Fine	No	No	No	No	Data transf. number	Yes
Yu [72]	No	Hypergraph part.	Fine	No	No	No	No	Cut weight	Yes
Zhang [90]	No	Lagrance MIP relaxation	Coarse	No	No	No	No	Data access cost	No
Hsu [91]	No	–	Fine	No	No	No	No	Profiling-related metric	Yes
LeBeane [97]	No	Hypergraph part.	Fine	No	No	No	No	Skew factor	Yes
Lan [89]	No	Clustering-based PSO search	Fine	No	No	No	No	Volatility AMA, hurst distance	Yes
BitDew [94]	No		Fine	Yes	Yes	No	Yes	Data dep. repl., fault tol.	Yes
Kayoor [99]	No	Hypergraph part.	Coarse	No	No	No	Yes	Avg. query span	Yes
Kosar [81]	Yes		Fine	Yes	Yes	No	Yes		Yes
Scalia [86]	No	Multi-dimensional Knapsack problem	Fine	No	Yes	Yes	No	Storage cost	Yes
SWORD [98]	Yes	Graph partition	Fine		No		Yes	Conflicting transactions	Yes

information—whether additional knowledge is captured which could enable the production of better data placement solutions. An overview of the evaluation based on these criteria can be observed in comparison Table 4. First of all, we can clearly see that there is no approach that covers all the criteria considered. Three approaches (Yuan et al. [78], BitDew [94] and Kosar [81]) can be distinguished, considered also as complementary to each other. However, only in [78] a suitable optimisation/scheduling algorithm for data placement has been realised.

Considering now each criterion in isolation, we can observe in Table 4 that very few approaches consider the existence of a fixed or semi-fixed location of data sets. Further, such approaches seem to prescribe a fixed a-priori solution to the data placement problem which can lead to a sub-optimal solution. Especially as optimisation opportunities are lost in sight of more flexible semi-fixed location constraints. For instance, fixing the placement of a dataset to a certain DC might be sub-optimal in case that multiple DCs in the same location exist.

Three main classes of data placement optimisation techniques can be observed: (i) meta-search (like PSO)/genetic programming) to more flexibly inspect the available solution space and efficiently find a near-optimal solution; (ii) hierarchical partition algorithms (based on BEA) that attempt to group data recursively based on data dependencies either to reduce the number or the cost of data transfers. BEA is used as the baseline for many of these algorithms. BEA also supports dynamicity. In particular, new data sets are handled by initially encoding them in a reduced table-based form before applying the BEA. After the initial solution is found, the modification can be done by adding cluster/VM capacity constraints into the model. (iii) a Big Data placement problem can also be encoded via a hypergraph. Here, nodes are data and machines while hyper-edges attempt to connect them together. Through such modelling, traditional or extended hypergraph partitioning techniques can be applied to find the best possible partitions. There can be a trade-off between different parameters or metrics that should be explored by all the data placement algorithms irrespectively of the constraint solving technique used. However, such a trade-off is not usually explored as in most cases only one metric is employed for optimisation.

Granularity constitutes the criterion with less versatility as most of the approaches have selected a fine-grained approach for data-to-resource mapping, which is suitable for the Cloud ecosystem.

The real-world applications are dynamic and can have varying load at different points of time. Furthermore, applications can produce additional data which can be used for next computation steps. Thus, data placement should be a continuous process to validate decisions taken at different points in time. However, most approaches in data placement, focus mainly on the initial positioning of Big Data and do not interfere with the actual runtime of the applications.

There seems also to exist a dependency between this criterion and the fixed data sets one. The majority of the proposed approaches satisfying this criterion also satisfy the fixed data set one. This looks like a logical outcome as dynamicity is highly correlated to the need to better handle some inherent data characteristics. Further, a large volume of intermediate data can also have a certain gravity effect that could resemble the one concerning fixed data.

The multi-application criterion is not supported at all. This can be due to the following facts: (i) multi-application support can increase the complexity and the size of the problem; (ii) it can also impact the solution quality and solution time which can be undesirable especially for approaches that already supply sub-optimal solutions.

Only the approach in [78] caters for data growth via reserving additional space in already allocated nodes based on statically specified margins. However, such an approach is static in nature and faces two unmet challenges: the support for dynamic data growth monitoring, suitable especially in cases where data can grow fast, and dynamic storage capacity determination, through, e.g., data growth prediction, for better supporting proactive data allocation. However, if we consider all dynamicity criteria together, we can nominate the approach in [78] as the one with the highest level of dynamicity, which is another indication of why it can be considered as prominent.

Data replication has been widely researched in the context of distributed systems but has not been extensively employed in data placement. Thus, we do believe that there exists a research gap here. Especially as those few approaches (such as SWORD [98], Kosar [81], Kayoor [99], BitDew [94]) that do support replication still lack suitable details or rely on very simple policies driven by user input.

We can observe that the minimisation of data transfer number or cost is a well-accepted optimisation criterion. Furthermore, data partitioning related criteria, such as *skew factor* and *cut weight*, have been mostly employed in the hypergraphs based methods. In some cases, we can also see multiple criteria to be considered which are: (i) either reduced to an overall one; (ii) not handled through any kind of optimisation but just considered in terms of policies that should be enforced. In overall, we are not impressed by the performance of the state-of-the-art in this comparison criterion. So, there is a huge room for potential improvement here.

Finally, many of the methods also consider additional input to achieve a better solution. The most common extra information that is exploited is data access patterns and nodes (VMs or PMs) profiling to, e.g., inspect their (data) processing speed. However, while both are important, usually only one from these two is exploited in these methods.

Lessons learned and future research directions

To conclude our survey, in this section we will discuss the issues of the current state-of-the-art and the research gaps or opportunities related to data storage and placement. Further, we also supply research directions towards a complete DLMS system in the Big Data-Cloud ecosystem.

Data lifecycle management

Challenges and issues

This subsection refers to how the discussed data storage and placement challenges can be combined and viewed from the perspective of a holistic DLMS of the future. Such a DLMS should be able to cope with the optimal data storage and placement in a way that considers the Big Data processing required, along with the functional and non-functional variability space of the given Cloud resources at hand, in each application scenario. It implies the ability to consider both private and public Clouds, offered by one or several Cloud vendors, according to the specifics of each use cases, while making the appropriate decisions on how the data should be stored, placed, processed and eventually managed.

Just considering the cross-Cloud application deployment for fully exploiting the benefits of the Cloud paradigm hinders the important challenge of data-awareness. This data-awareness refers to the need to support an application deployment process that considers the locations of data sources, their volume and velocity characteristics, as well as any security and privacy constraints applicable. Of course, from the DLM perspective, this means that there should also be a consideration of the dependencies between application components and all data sources. This has the reasonable implication that the components requiring frequent accesses to data artefacts, found at rest in certain data stores, cannot be placed in a different Cloud or even in a certain physical and network distance from the actual storage location. If such aspects are ignored then application performance certainly degrades, as expensive data migrations may incur while legislation conformance issues might be applicable.

Future research directions

Among the most prominent research directions, we highlighted the design and implementation of a holistic DLMS, able to cope with all of the above-mentioned aspects on the data management, while employing the appropriate strategies for benefiting from the multi-Cloud paradigm. It is important to note that data placement in virtualized resources is generally subjected to long-term decisions as any potential data migrations generally incur immense costs which may be amplified by data gravity aspects that may result in subsequent changes in the application placement. Based on this, we consider the following aspects that should sketch the main functionality of the DLMS of the future that is able to cope with Big Data management and processing by really taking advantage of the abundance of resources in the Cloud computing world:

- Use of advanced modelling techniques that consider metadata schemas for setting the scope of truly exploitable data modelling artefacts. It refers to managing the modelling task in a way that covers the description of all V's (e.g. velocity, volume, value, variety, and veracity) in the characteristics of Big Data to be processed. The proper and multi-dimensional data modelling will allow for an adequate description of the data placement problem.
- Perform optimal data placement across multiple Cloud resources based on the data modelling and user-defined goals, requirements and constraints.
- Use of efficiently distributed monitoring functionalities for observing the status of the Big Data stored or processed and detect any migration or reconfiguration opportunities.
- Employ the appropriate replication, fail-over and backup techniques by considering and exploiting at the same time the already offered functionalities by public Cloud providers.
- According to such opportunities, continuously make reconfiguration and migration decisions by consistently considering the real penalty for the overall application reconfiguration, always in sight of the user constraints, goals and requirements that should drive the configuration of computational resources and the scheduling of application tasks.
- Design and implement security policies in order to guarantee that certain regulations (e.g., General Data Protection Regulation) are constantly and firmly respected (e.g., data artefacts should not be stored or processed outside the European Union) while at the same time the available Cloud providers' offerings are exploited according to the data owners' privacy needs (e.g., exploit the data sanitization service when migrating or just removing data from a certain Cloud provider).

Data storage

In this section, we highlight the challenges for holistic data lifecycle management with respect to both the current DBMS and DFS systems and propose future research directions to overcome such challenges.

Challenges and issues

In the recent decade, the DBMS landscape has significantly evolved with respect to the data models and supported non-functional features, driven by Big Data and the related requirements of Big Data applications (see "Non-functional data management features" section). The resulting heterogeneous DBMS landscape provides a lot of new opportunities for Big Data management while it simultaneously imposes new challenges as well. The variety of data models offers domain-specific solutions for different kinds of data structures. Yet, the vast number of existing DBMSs per data model leads to a complex DBMS selection process. Hereby, functional features of potential DBMSs need to be carefully evaluated (e.g., NoSQL DBMSs do not offer a common query interface even within the same data model). For the non-functional features, the decision process is twofold: (i) a qualitative analysis (as carried out in "Comparison of selected DBMSs" section) should be conducted to narrow down the potential DBMSs; (ii) quantitative evaluations should be performed over the major non-functional features based on existing evaluation frameworks.

While collecting data from many distributed and diverse data sources is a challenge [8] modern Big Data applications are typically built upon multiple different data structures. Consequently, current DBMSs cater for domain-specific data structures due to the variety of data models supported, (as shown in our analysis Table 2). However, exploiting the variety of data models typically leads to the integration of multiple different DBMSs in modern Big Data applications. Consequently, the operation of a DBMS needs to be abstracted to ease the integration of different DBMSs into Big Data applications and to fully exploit the required features (such as scalability or elasticity). Hereby, research approaches in Cloud-based application orchestration can be exploited [101, 102]. While the current DBMS landscape already moves towards the Big Data domain, the optimal operation of large-scale or even geo-distributed DBMSs still remains a challenge as the non-functional features significantly differ for different DBMSs (especially by using Cloud resources [42, 61, 103]).

In general, DFS provides scalability, network transparency, fault tolerance, concurrent data (I/O) access, and data protection [104]. It is worth noting that in Big Data domain, the scalability must be achieved without increasing the degree of replication of stored data (particularly for the Cloud ecosystem while combined with the private/local data storage systems). The storage system must increase user data availability but not the overheads. While resource sharing is a complex task and the severity can increase many-folds while managing the Big Data. In today's Cloud ecosystem, we lack a single/unified model that offers a single interface to connect multiple Cloud-based storage models (such as Amazon S3 objects) and DFSs. Apart from that, the synchronization in DFS is also a well-known issue and as the degree of data access concurrency is increasing, synchronization could certainly be a performance bottleneck. Moreover, in some cases, it has also been observed that the performance of DFSs is low compared to the local file systems [105, 106]. Furthermore, network transparency is also a crucial process related to the performance, especially while handling Big Data (because now the Big Data is distributed across multiple Clouds). Although most DFSs uses transmission control protocol or user datagram protocol during the communication process, however, a smarter way needs to be devised. In DFS, the fault-tolerance is achieved by lineage, checkpoint, and replicating metadata (and data objects) [104]. While the state-less based DFSs are having fewer overheads regarding managing the file states while reconnecting after failures, the state-full approach is also in use. For DFSs, the failure must be handled very fast and seamlessly across the Big Data management infrastructure. On the other side, there is no well-accepted approach to data access optimization methods. The methods such as data locality, multi-level caches are used case by case. Finally, securing the data in the DFS-Cloud ecosystem is a challenge due to the interconnection of so many diverse hardware as well as software components.

Future research directions

To address the identified challenges for the data storage in Big Data lifecycle management, novel Big Data-centric evaluations are required that ease the selection and operation of large-scale DBMS.

- The growing domain of hybrid transaction/analytical processing workloads needs to be considered for the existing data models. Moreover, comparable benchmarks for different data models need to be established [107] and qualitative evaluations need to be performed across all data model domains as well.
- To select an optimal combination of a distributed DBMS and Cloud resources, evaluation frameworks across different DBMS, Cloud resource and workload domains are required [108]. Such frameworks ease the DBMS selection and operation for Big Data lifecycle management.
- Holistic DBMS evaluation frameworks are required to enable the qualitative analysis across all non-functional features in a comparable manner. In order to achieve this, frameworks need to support complex DBMS adaptation scenarios, including scaling and failure injection.
- DBMS adaptation strategies need to be derived and integrated into the orchestration frameworks to enable the automated operation (to cope with workload fluctuations) of a distributed DBMS.
- Qualitative DBMS selection guidelines need to be extended with respect to operational and adaptation features of current DBMS (i.e., support for orchestration frameworks to enable automated operation and adaptation and the integration support into Big Data frameworks).

Similar to the above research directions for DBMSs, we also mention below the research directions for DFSs.

- For efficient, resource sharing among multiple Cloud service providers/components, a single/unified interface must handle the complex issues, such as seamless

workload distribution, improved data access experience and faster read-write synchronizations, together with the increased level of data serialization for DFSs.
- We also advocate for using smarter replica-assignment policies to achieve better workload balance and efficient storage space management.
- To counter the synchronization issue in DFSs, a generic solution could be to cache the data in the client or in the local server's side, but such an approach can become the bottleneck for the Big Data management scenario as well. Thus, exploratory research must be done in this direction.
- As the data diversity and the networks heterogeneity is increasing, an abstract communication layer must be in place to address the issue of network transparency. Such abstraction can handle different types of communications easily and efficiently.
- The standard security mechanisms are in place (such as ACLs) for data security. However, after the Cloudification of the file system, the data become more vulnerable due to the interconnection of diverse distributed, heterogeneous computing components. Thus, proper security measures must be built-in features of tomorrow's DFSs.

Data placement

The following data placement challenges and corresponding research directions are in line with our analysis in "Comparative evaluation" section.

Challenges and issues

Fixed data set size We have observed data placement methods able to fix the location of data sets based on respective (privacy) regulations, laws or user requirements. Such requirements indicate that data placement should be restrained within a certain country, sets of countries or even continents. However, this kind of semi-fixed constraints is handled in a rather static way by already pre-selecting the right place for such data sets.

Constraint solving Exhaustive solution techniques are efficient to reach optimal solutions but suffer from scalability issues and higher execution time (especially for medium/big-sized problem instances). On the other hand, meta-heuristics (such as PSO) seems more promising as they can produce near-optimal solutions faster by also achieving better scalability. However, they need proper configuration and modelling which can be a time-consuming task while it is not always guaranteed that near-optimal solutions can be produced.

Granularity Most of the evaluated methods support a fine-grained approach for dataset placement. However, all such methods consider that resources are fixed in number. Such assumptions are inflexible in the sight of the following issues: (i) a gradual data growth can saturate the resources assigned to data. In fact, a whole private storage infrastructure could be saturated for this reason; (ii) data should be flexibly (re-)partitioned to tackle the workload variability.

Multiple applications Only three from the evaluated methods (see Table 4) can handle multiple applications but also in a very limited fashion. Such handling is challenging, especially when different applications are assorted with conflicting requirements. It must also be dynamic due to the changes brought by application execution as well as other factors (e.g., application requirement and Cloud infrastructure changes).

Data growth Data sets can grow over time. Only one method [78] in the previous analysis is able to handle the data size change. It employs a threshold-based approach to check when data needs to be moved or when resources are adequate for storing the data to also handle their growth. However, no detailed explanation is supplied concerning how the threshold is computed.

Data replication It is usually challenging to find the best possible trade-off between cost and replication degree to enable cost-effective data replication.

Optimisation criteria Data transfer and replication management is a complex process [109] due to the completely distributed nature of the Cloud ecosystem. It further gets complicated due to the unequal data access speed. Data transfer number or cost is a well-accepted criterion for optimising data placement. However, it can be also quite restrictive. First, as there can be cases where both of these two metrics need to be considered. For instance, suppose that we need to place two datasets, initially situated in one VM, to other VMs as this VM will become soon unavailable. If we just consider the transfer number, this can lead to the situation where the movement is performed in an arbitrary way even migrating data to another DC while there is certainly a place in the current one. In the opposite direction, there can be cases where cost could be minimised but this could lead to increasing the number of transfers which could impact application performance. Second, data placement has been mainly seen in an isolated manner without examining user requirements. However, it can greatly affect application performance and cost.

Additional information Apart from extracting data access patterns and node profiles, we believe that more information is needed for a better data placement solution.

Future research directions

- Fixed data set size: To guarantee the true, optimal satisfaction of the user requirements and optimisation objectives, we suggest *the use of semi-fixed constraints in a more suitable and flexible manner as a respective non-static part of the location-aware optimisation problem to be solved.*
- Constraint solving: We propose *the use of hybrid approaches* (i.e., combining exhaustive and meta-search heuristic techniques) so as to rapidly get (within an acceptable and practically employable execution time) optimal or near-optimal results in a scalable fashion. For instance, constraint programming could be combined with local search. The first could be used to find a good initial solution, while the latter could be used for neighbourhood search to find a better result. In addition, it might be possible that *a different and more scalable modelling of the optimisation problem* could enable to run standard exhaustive solution techniques even with medium-

sized problem instances. Finally, *solution learning from history* could be adopted to fix parts of the optimisation problem and thus substantially reduce the solution space to be examined.

- Granularity: There is a need for *dynamic approaches for data placement* which do take into account the *workload fluctuation* and the *data growth* to both partition data as well as optimally place them in a set of resources with a *size that is dynamically identified*.
- Multiple applications: To handle applications conflicting requirements and the dynamicity of context (e.g., change of infrastructure, application requirements), *different techniques to solve the (combined) optimisation problem* are required. First, *soft constraints* could be used to solve this problem, even if it is over-constrained (e.g., producing a solution that violates the least number of these preferences). Next, we could *prioritise the applications and/or their tasks*. Third, *distributed solving techniques* could be used to produce application-specific optimisation problems of reduced complexity. This would require a *transformation of the overall problem into sub-problems which retains as much as possible the main constraints and requirements of each relevant application*. Finally, complementary to these distributed solving techniques, the *measure of replication* could also be employed. By using such a measure, we enable each application to operate over its own copy of the data originally shared. This could actually enable to *have complete independence of applications* which would then allow us to solve data placement individually for each of these applications.
- Data growth: There is a need to employ a *more sophisticated approach* which exploits the *data (execution) history* as well as *data size prediction and data (type) similarity techniques* to solve the data growth issue. Similarity can be learned by knowing *the context of data* (e.g., by assuming the same context has been employed for similar data over time by multiple users), while *statistical methods* can predict the data growth. Such an approach can also be used for new data sets for which no prior knowledge exists (known as the *cold-start problem*).
- Data replication: For data replication, we suggest to dynamically *compute the replication degree by considering the application size, data size, data access pattern, data growth rate, user requirements, and the capabilities of Cloud services*. Such a solution could also rely on *a weight calculation method* for the determination of the relative importance of each of these factors.
- Optimisation criteria: An interesting research direction compiles into *exploring ways via data placement and task scheduling could be either solved in conjunction or in a clever but independent way such that they do take into account the same set of (high-level) user requirements*. This could lead to producing solutions which are in concert and also optimal according to both aspects of data and computation.
- Additional information: We advocate that the additional information required to be collected or derived include: (i) *co-locating frequently accessing tasks and data*; (ii) *exploiting data dependencies to have effective data partitioning*. A similar approach is employed by Wang et al. [83] where data are grouped together at a finer granularity. There are also precautions in not storing different data blocks from the same data in the same node; (iii) *data variability* data can be of different forms. Each form might require a different machine configuration for optimal storage and processing.

In this case, *profiling should be extended to also capture this kind of machine performance variation* which could be quite beneficial for *more data-form-focused placement*. In fact, we see that whole approaches are dedicated to dealing with different data forms. For instance, graph analytics-oriented data placement algorithms exploit the fact that data are stored in the form of graphs to more effectively select the right techniques and algorithms for solving the data placement problem. While special-purpose approaches might be suitable for different data forms, they are not the right choice for handling different kinds of data. As such, we believe that an important future direction should be the ability to *more optimally handle data of multiple forms* to enhance the applicability of a data placement algorithm and make it suitable for handling different kinds of applications instead of a single one.

Concluding remarks

The primary aim of this survey is to provide a holistic overview of the state of the art related to both data storage and placement in the Cloud ecosystem. We acknowledge that there do exist some surveys on various aspects of Big Data, which focus on the functional aspect and mainly on Big Data storage issues. However, this survey plays a complementary role with respect to them. In particular, we cover multiple parts of the Big Data management architecture (such as DLM, data storage systems, data placement techniques), which were neglected in the other surveys, under the prism of non-functional properties. Further, our contribution to Big Data placement is quite unique. In addition, the in-depth analysis of each main article section is covered by a well-designed set of evaluation criteria. Such an analysis also assists in a better categorization of the respective approaches (or technologies, involved in each part).

Our survey enables readers to better understand which solution could be utilized under which non-functional requirements. Thus, assisting towards the construction of user-specific Big Data management systems according to the non-functional requirements posted. Subsequently, we have described relevant challenges that can pave the way for the proper evolution of such systems in the future. Each challenge prescribed in "Lessons learned and future research directions" section has been drawn from the conducted analysis. Lastly, we have supplied a set of interesting and emerging future research work directions concerning both the functionalities related to the Big Data management (i.e., Big Data storage and placement), as well as the Big Data lifecycle management as a whole, in order to address the identified challenges.

Abbreviations

ACL: access control list; BDaaS: Big Data-as-a-service; BEA: bond energy algorithm; BI: business intelligence; CDM: cognitive data management; DaaS: data-as-a-service; DBaaS: database-as-a-service; DCs: data centers; DDBMS: distributed database management system; DFS: distributed file system; DLMS: data lifecycle management system; DMS: data management system; HA: high availability; HDFS: Hadoop distributed file system; IoT: internet-of-thing; NFS: Network File System; PaaS: platform-as-a-service; PSO: particle swarm optimization; PVFS: parallel virtual file system; QoS: quality of service; SLR: systematic literature review; STaaS: storage-as-a-service.

Authors' contributions

"Introduction" is contributed by SM, DS, KK and YV; "Data lifecycle management (DLM)" is contributed by SM, KK and YV; "Methodology" is contributed by KK and SM; "Non-functional data management features" is contributed by DS and SM; "Data storage systems" is contributed by DS, SM and YV; "Data placement techniques" is contributed by KK and SM; and "Lessons learned and future research directions" is contributed by YV, SM, DS, KK. All authors read and approved the final manuscript.

Author details
[1] Simula Research Laboratory, 1325 Lysaker, Norway. [2] Ulm University, Ulm, Germany. [3] ICS-FORTH, Heraklion, Crete, Greece. [4] Institute of Communication and Computer Systems (ICCS), 9 Iroon Polytechniou Str., Athens, Greece.

Acknowledgements
The research leading to this survey paper has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 731664. The authors would like to thank the partners of the MELODIC project (http://www.melodic.cloud/) for their valuable advices and comments.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
Not applicable.

Funding
This work is generously supported by the Melodic project (Grant Number 731664) of the European Union H2020 program.

Publisher's Note

Received: 4 October 2018 Accepted: 22 January 2019
Published online: 11 February 2019

References
1. Khan N, Yaqoob I, Hashem IAT, et al. Big data: survey, technologies, opportunities, and challenges. Sci World J. 2014;2014:712826.
2. Kaisler S, Armour F, Espinosa JA, Money W. Big data: issues and challenges moving forward. In: System sciences (HICSS), 2013 46th Hawaii international conference on, IEEE. 2013. pp. 995–1004.
3. Zheng Z, Zhu J, Lyu MR. Service-generated big data and big data-as-a-service: an overview. In: Big Data (BigData Congress), 2013 IEEE international congress on, IEEE. 2013. pp. 403–10.
4. Chen M, Mao S, Liu Y. Big data: a survey. Mob Netw Appl. 2014;19(2):171–209.
5. Inukollu VN, Arsi S, Ravuri SR. Security issues associated with big data in cloud computing. Int J Netw Secur Appl. 2014;6(3):45.
6. Wang C, Wang Q, Ren K, Lou W. Privacy-preserving public auditing for data storage security in cloud computing. In: Infocom, 2010 proceedings IEEE, IEEE. 2010. pp. 1–9.
7. Chaudhuri S. What next?: a half-dozen data management research goals for big data and the cloud. In: PODS, Scottsdale, AZ, USA. 2012. pp. 1–4.
8. Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E. Deep learning applications and challenges in big data analytics. J Big Data. 2015;2(1):1.
9. Verma D. Supporting service level agreements on IP networks. Indianapolis: Macmillan Technical Publishing; 1999.
10. Sakr S, Liu A, Batista DM, Alomari M. A survey of large scale data management approaches in cloud environments. IEEE Commun Surv Tutor. 2011;13(3):311–36.
11. Wu L, Yuan L, You J. Survey of large-scale data management systems for big data applications. J Comput Sci Technol. 2015;30(1):163.
12. Oussous A, Benjelloun FZ, Lahcen AA, Belfkih S. Big data technologies: a survey. J King Saud Univ Comput Inf Sci. 2017;30(4):431–48.
13. Grolinger K, Higashino WA, Tiwari A, Capretz MA. Data management in cloud environments: NoSQL and NewSQL data stores. J Cloud Comput Adv Syst Appl. 2013;2(1):22.
14. Zhang H, Chen G, Ooi BC, Tan KL, Zhang M. In-memory big data management and processing: a survey. IEEE Trans Knowl Data Eng. 2015;27(7):1920–48.
15. Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Khan SU. The rise of "big data" on cloud computing: review and open research issues. Inf Syst. 2015;47:98–115.
16. Ball A. Review of data management lifecycle models. Bath: University of Bath; 2012.
17. Demchenko Y, de Laat C, Membrey P. Defining architecture components of the big data ecosystem. In: International conference on collaboration technologies and systems. 2014. pp. 104–12.
18. Pääkkönen P, Pakkala D. Reference architecture and classification of technologies, products and services for big data systems. Big Data Res. 2015;2(4):166–86.
19. NBD-PWG. NIST big data interoperability framework: volume 2, big data taxonomies. Tech. rep., NIST, USA 2015. Special Publication 1500-2.
20. Organisation for Economic Co-operation and Development. Data-driven innovation: big data for growth and well-being. Paris: OECD Publishing; 2015.
21. Kaufmann M. Towards a reference model for big data management. Research report, University of Hagen. 2016. Retrieved from https://ub-deposit.fernuni-hagen.de/receive/mir_mods_00000583. Retrieved 15 July 2016.
22. Höfer C, Karagiannis G. Cloud computing services: taxonomy and comparison. J Internet Serv Appl. 2011;2(2):81–94.
23. Baru C, Moore R, Rajasekar A, Wan M. The sdsc storage resource broker. In: CASCON first decade high impact papers, IBM Corp.; 2010. pp. 189–200.

24. Chasen JM, Wyman CN. System and method of managing metadata data 2004. US Patent 6,760,721.
25. Gómez A, Merseguer J, Di Nitto E, Tamburri DA. Towards a uml profile for data intensive applications. In: Proceedings of the 2nd international workshop on quality-aware DevOps, ACM. 2016. pp. 18–23.
26. Verginadis Y, Pationiotakis I, Mentzas G. Metadata schema for data-aware multi-cloud computing. In: Proceedings of the 14th international conference on INnovations in Intelligent SysTems and Applications (INISTA). IEEE. 2018.
27. Binz T, Breitenbücher U, Kopp O, Leymann F. Tosca: portable automated deployment and management of cloud applications. In: Advanced web services. Springer; 2014. pp. 527–49.
28. Kritikos K, Domaschka J, Rossini A. Srl: a scalability rule language for multi-cloud environments. In: Cloud computing technology and science (CloudCom), 2014 IEEE 6th international conference on, IEEE. 2014. pp. 1–9.
29. Rossini A, Kritikos K, Nikolov N, Domaschka J, Griesinger F, Seybold D, Romero D, Orzechowski M, Kapitsaki G, Achilleos A. The cloud application modelling and execution language (camel). Tech. rep., Universität Ulm 2017.
30. Das S, Nishimura S, Agrawal D, El Abbadi A. Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration. Proc VLDB Endow. 2011;4(8):494–505.
31. Lu C, Alvarez GA, Wilkes J. Aqueduct: online data migration with performance guarantees. In: Proceedings of the 1st USENIX conference on file and storage technologies, FAST '02. USENIX Association 2002.
32. Stonebraker M, Devine R, Kornacker M, Litwin W, Pfeffer A, Sah A, Staelin C. An economic paradigm for query processing and data migration in mariposa. In: Parallel and distributed information systems, 1994., proceedings of the third international conference on, IEEE. 1994. pp. 58–67.
33. Brubeck DW, Rowe LA. Hierarchical storage management in a distributed VOD system. IEEE Multimedia. 1996;3(3):37–47.
34. Kitchenham BA, Pfleeger SL, Pickard LM, Jones PW, Hoaglin DC, Emam KE, Rosenberg J. Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng. 2002;28(8):721–34.
35. Gessert F, Wingerath W, Friedrich S, Ritter N. NoSQL database systems: a survey and decision guidance. Comput Sci Res Dev. 2017;32(3–4):353–65.
36. Sakr S. Cloud-hosted databases: technologies, challenges and opportunities. Clust Comput. 2014;17(2):487–502.
37. Cattell R. Scalable SQL and NoSQL data stores. Acm Sigmod Rec. 2011;39(4):12–27.
38. Gray J. Database and transaction processing performance handbook. In: The benchmark handbook for database and transaction systems. 2nd ed. Digital Equipment Corp. 1993.
39. Traeger A, Zadok E, Joukov N, Wright CP. A nine year study of file system and storage benchmarking. ACM Trans Storage. 2008;4(2):5.
40. Agrawal D, El Abbadi A, Das S, Elmore AJ. Database scalability, elasticity, and autonomy in the cloud. In: International conference on database systems for advanced applications. Springer. 2011. pp. 2–15.
41. Séguin C, Le Mahec G, Depardon B. Towards elasticity in distributed file systems. In: Cluster, cloud and grid computing (CCGrid), 2015 15th IEEE/ACM international symposium on, IEEE. 2015. pp. 1047–56.
42. Seybold D, Wagner N, Erb B, Domaschka J. Is elasticity of scalable databases a myth? In: Big Data (Big Data), 2016 IEEE international conference on, IEEE. 2016. pp. 2827–36.
43. Gilbert S, Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. Acm Sigact News. 2002;33(2):51–9.
44. Bermbach D, Kuhlenkamp J. Consistency in distributed storage systems. In: Networked systems. Springer. 2013. pp. 175–89.
45. Lakshman A, Malik P. Cassandra: a decentralized structured storage system. ACM SIGOPS Oper Syst Rev. 2010;44(2):35–40.
46. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE. Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst. 2008;26(2):4.
47. Pavlo A, Aslett M. What's really new with newsql? ACM Sigmod Rec. 2016;45(2):45–55.
48. Corbellini A, Mateos C, Zunino A, Godoy D, Schiaffino S. Persisting big-data: the NoSQL landscape. Inf Syst. 2017;63:1–23.
49. Davoudian A, Chen L, Liu M. A survey on NoSQL stores. ACM Comput Surv. 2018;51(2):40.
50. Jensen SK, Pedersen TB, Thomsen C. Time series management systems: a survey. IEEE Trans Knowl Data Eng. 2017;29(11):2581–600.
51. Bader A, Kopp O, Falkenthal M. Survey and comparison of open source time series databases. In: BTW (Workshops). 2017. pp. 249–68.
52. Abadi JD. Data management in the cloud: limitations and opportunities. IEEE Data Eng Bull. 2009;32:3–12.
53. Pritchett D. Base: an acid alternative. Queue. 2008;6(3):48–55.
54. Codd EF. Extending the database relational model to capture more meaning. ACM Trans Database Syst. 1979;4(4):397–434.
55. Aslett M. How will the database incumbents respond to nosql and newsql. The San Francisco. 2011;451:1–5.
56. Sadalage PJ, Fowler M. NoSQL distilled. 2012. ISBN-10 321826620
57. Seybold D, Hauser CB, Volpert S, Domaschka J. Gibbon: an availability evaluation framework for distributed databases. In: OTM confederated international conferences "On the Move to Meaningful Internet Systems". Springer. 2017. pp. 31–49.
58. Seybold D, Domaschka J. Is distributed database evaluation cloud-ready? In: Advances in databases and information systems. Springer. 2017. pp. 100–8.
59. Barahmand S, Ghandeharizadeh S. BG: a benchmark to evaluate interactive social networking actions. In: CIDR. Citeseer. 2013.
60. Cooper BF., Silberstein A, Tam E, Ramakrishnan R, Sears R. Benchmarking cloud serving systems with ycsb. In: Proceedings of the 1st ACM symposium on Cloud computing, ACM. 2010. pp. 143–54.
61. Kuhlenkamp J, Klems M, Röss O. Benchmarking scalability and elasticity of distributed database systems. Proc VLDB Endow. 2014;7(12):1219–30.
62. Bermbach D, Tai S. Benchmarking eventual consistency: lessons learned from long-term experimental studies. In: Cloud engineering (IC2E), 2014 IEEE international conference on, IEEE. 2014. pp. 47–56.

63. Domaschka J, Hauser CB, Erb B. Reliability and availability properties of distributed database systems. In: Enterprise distributed object computing conference (EDOC), 2014 IEEE 18th international, IEEE. 2014. pp. 226–33.
64. Brewer E. Cap twelve years later: How the "rules" have changed. Computer. 2012;45(2):23–9.
65. Klems M, Bermbach D, Weinert R. A runtime quality measurement framework for cloud database service systems. In: Quality of information and communications technology (QUATIC), 2012 eighth international conference on the, IEEE. 2012. pp. 38–46.
66. Abadi D, Agrawal R, Ailamaki A, Balazinska M, Bernstein PA, Carey MJ, Chaudhuri S, Chaudhuri S, Dean J, Doan A. The beckman report on database research. Commun ACM. 2016;59(2):92–9.
67. Group NBDPW, et al. Nist big data interoperability framework. Special Publication 2015. pp. 1500–6.
68. Kachele S, Spann C, Hauck FJ, Domaschka J. Beyond iaas and paas: an extended cloud taxonomy for computation, storage and networking. In: Utility and cloud computing (UCC), 2013 IEEE/ACM 6th international conference on, IEEE. 2013. pp. 75–82.
69. Levy E, Silberschatz A. Distributed file systems: concepts and examples. ACM Comput Surv. 1990;22(4):321–74.
70. Muthitacharoen A, Morris R, Gil TM, Chen B. Ivy: a read/write peer-to-peer file system. ACM SIGOPS Oper Syst Rev. 2002;36(SI):31–44.
71. Ross RB, Thakur R, et al. PVFS: a parallel file system for Linux clusters. In: Proceedings of the 4th annual Linux showcase and conference. 2000. pp. 391–430.
72. Yu B, Pan J. Sketch-based data placement among geo-distributed datacenters for cloud storages. In: INFOCOM, San Francisco: IEEE. 2016. pp. 1–9.
73. Greene WS, Robertson JA. Method and system for managing partitioned data resources. 2005. US Patent 6,922,685.
74. Greenberg A, Hamilton J, Maltz DA, Patel P. The cost of a cloud: research problems in data center networks. ACM SIGCOMM Comput Commun Rev. 2008;39(1):68–73.
75. Hardavellas N, Ferdman M, Falsafi B, Ailamaki A. Reactive nuca: near-optimal block placement and replication in distributed caches. ACM SIGARCH Comput Archit News. 2009;37(3):184–95.
76. Kosar T, Livny M. Stork: making data placement a first class citizen in the grid. In: Distributed computing systems, 2004. Proceedings. 24th international conference on, IEEE. 2004. pp. 342–9.
77. Xie T. Sea: a striping-based energy-aware strategy for data placement in raid-structured storage systems. IEEE Trans Comput. 2008;57(6):748–61.
78. Yuan D, Yang Y, Liu X, Chen J. A data placement strategy in scientific cloud workflows. Future Gener Comput Syst. 2010;26(8):1200–14.
79. Doraimani S, Iamnitchi A. File grouping for scientific data management: lessons from experimenting with real traces. In: Proceedings of the 17th international symposium on High performance distributed computing, ACM. 2008. pp. 153–64.
80. Cope JM, Trebon N, Tufo HM, Beckman P. Robust data placement in urgent computing environments. In: Parallel & distributed processing, 2009. IPDPS 2009. IEEE international symposium on, IEEE. 2009. pp. 1–13.
81. Kosar T, Livny M. A framework for reliable and efficient data placement in distributed computing systems. J Parallel Distrib Comput. 2005;65(10):1146–57.
82. Bell DA. Difficult data placement problems. Comput J. 1984;27(4):315–20.
83. Wang J, Shang P, Yin J. Draw: a new data-grouping-aware data placement scheme for data intensive applications with interest locality. IEEE Trans Magnetic. 2012;49(6):2514–20.
84. McCormick W, Schweitzer P, White T. Problem decomposition and data reorganisation by a clustering technique. Oper Res. 1972;20:993–1009.
85. Ebrahimi M, Mohan A, Kashlev A, Lu S. BDAP: a Big Data placement strategy for cloud-based scientific workflows. In: BigDataService, IEEE computer society. 2015. pp. 105–14.
86. Papaioannou TG, Bonvin N, Aberer K. Scalia: an adaptive scheme for efficient multi-cloud storage. In: Proceedings of the international conference on high performance computing, networking, storage and analysis. IEEE Computer Society Press. 2012. p. 20.
87. Er-Dun Z, Yong-Qiang Q, Xing-Xing X, Yi C. A data placement strategy based on genetic algorithm for scientific workflows. In: CIS, IEEE computer society. 2012. pp. 146–9.
88. Rafique A, Van Landuyt D, Reniers V., Joosen W. Towards an adaptive middleware for efficient multi-cloud data storage. In: Proceedings of the 4th workshop on CrossCloud infrastructures & platforms, Crosscloud'17. 2017. pp. 1–6.
89. Lan K, Fong S, Song W, Vasilakos AV, Millham RC. Self-adaptive pre-processing methodology for big data stream mining in internet of things environmental sensor monitoring. Symmetry. 2017;9(10):244.
90. Zhang J, Chen J, Luo J, Song A. Efficient location-aware data placement for data-intensive applications in geo-distributed scientific data centers. Tsinghua Sci Technol. 2016;21(5):471–81.
91. Hsu CH, Slagter KD, Chung YC. Locality and loading aware virtual machine mapping techniques for optimizing communications in mapreduce applications. Future Gener Comput Syst. 2015;53:43–54.
92. Xu Q, Xu Z, Wang T. A data-placement strategy based on genetic algorithm in cloud computing. Int J Intell Sci. 2015;5(3):145–57.
93. Chervenak AL, Smith DE, Chen W, Deelman E. Integrating policy with scientific workflow management for data-intensive applications. In: 2012 SC companion: high performance computing, networking storage and analysis. 2012. pp. 140–9.
94. Fedak G, He H, Cappello F. Bitdew: a programmable environment for large-scale data management and distribution. In: 2008 SC—international conference for high performance computing, networking, storage and analysis. 2008. pp. 1–12.
95. Deelman E, Singh G, Su MH, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci Program. 2005;13(3):219–37.

96. Chervenak A, Deelman E, Foster I, Guy L, Hoschek W, Iamnitchi A, Kesselman C, Kunszt P, Ripeanu M, Schwartzkopf B, et al. Giggle: a framework for constructing scalable replica location services. In: Proceedings of the 2002 ACM/IEEE conference on supercomputing. IEEE computer society press. 2002. pp. 1–17.
97. LeBeane M, Song S, Panda R, Ryoo JH, John LK. Data partitioning strategies for graph workloads on heterogeneous clusters. In: SC, Austin: ACM; 2015. pp. 1–12.
98. Quamar A, Kumar KA, Deshpande A. Sword: scalable workload-aware data placement for transactional workloads. In: Proceedings of the 16th international conference on extending database technology, EDBT '13, ACM. 2013. pp. 430–41.
99. Kumar KA, Deshpande A, Khuller S. Data placement and replica selection for improving co-location in distributed environments. CoRR 2012. arXiv:1302.4168.
100. Catalyurek UV, Kaya K, Uçar B. Integrated data placement and task assignment for scientific workflows in clouds. In: Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing, New York, NY, USA: ACM; 2011. pp. 45–54.
101. Baur D, Seybold D, Griesinger F, Tsitsipas A, Hauser CB, Domaschka J. Cloud orchestration features: are tools fit for purpose? In: Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th international conference on, IEEE. 2015. pp. 95–101.
102. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J. Borg, omega, and kubernetes. Queue. 2016;14(1):10.
103. Schad J, Dittrich J, Quiané-Ruiz JA. Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proc VLDB Endow. 2010;3(1–2):460–71.
104. Thanh TD, Mohan S, Choi E, Kim S, Kim P. A taxonomy and survey on distributed file systems. In: Networked computing and advanced information management, 2008. NCM'08. Fourth international conference on, vol. 1, IEEE. 2008. pp. 144–9.
105. Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I. Disk-locality in datacenter computing considered irrelevant. In: HotOS. 2011. p. 12.
106. Nightingale EB, Chen PM, Flinn J. Speculative execution in a distributed file system. In: ACM SIGOPS operating systems review, vol. 39, ACM. 2005. pp. 191–205.
107. Coelho F, Paulo J, Vilaça R, Pereira J, Oliveira R. Htapbench: Hybrid transactional and analytical processing benchmark. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ACM; 2017. pp. 293–304.
108. Seybold D, Keppler M, Gründler D, Domaschka J. Mowgli: Finding your way in the DBMS jungle. In: Proceedings of the 2019 ACM/SPEC international conference on performance engineering. ACM. 2019.
109. Allen MS, Wolski R. The livny and plank-beck problems: studies in data movement on the computational grid. In: Supercomputing, 2003 ACM/IEEE conference, IEEE. 2003. pp. 43.