Melodic
Big data cloud

Multi-cloud Execution-ware for Large-scale Optimised Data-Intensive Computing

H2020-ICT-2016-2017
Leadership in Enabling and Industrial Technologies; Information and Communication Technologies

Grant Agreement No.:
731664

Duration:
1 December 2016 -
30 November 2019

www.melodic.cloud

Deliverable reference:
D2.5

Date:
18 June 2018

Responsible partner:
Simula Research Laboratory

Editor(s):
Feroz Zahid

Author(s):
Kyriakos Kritikos, Feroz Zahid, Somnath Mazumdar, Daniel Seybold, Yiannis Verginadis

Approved by:
Antonia Schwichtenberg

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/ D2.5 Report on data placement and migration methodologies.pdf

Title:

D2.5 Report on Data Placement and Migration Methodologies

Abstract:

This document presents methods, techniques, and technologies incorporated in the Melodic middleware for optimised Cross-Cloud data placements and migrations. The document is logically divided into two parts. The first part provides a review of the existing state-of-the-art data management techniques in the Cloud. The second part reports on the Data Life-cycle Management System (DLMS), developed as part of the Melodic Upperware. All data sources available to the applications deployed through Melodic are modelled and registered in the DLMS. The job of the DLMS is to manage data sources on behalf of the Melodic users throughout their life-cycle. This covers, when required, the selection of an appropriate location for the initial placement of the datasets on a Cloud based on the user-defined requirements and data storage costs in various Cloud systems, and subsequent data migrations based on the application and data deployment solutions calculated by the solvers. DLMS also assigns a *utility value* to all proposed deployment solutions calculated by the solvers to assist in the optimisation based on the current application topology and data source locations. The utility value represents the degree to which a solution is favourable by the DLMS, considering any data migrations required and impact on the application performance by the prescribed placement of application components and data sources. The DLMS computes utility value based on historical data access patterns reflecting affinities between application components and data sources, dataset characteristics, average network latencies and throughput between data centres, Cloud provider costs, and predictions from past DLMS decisions, as implemented through the DLMS algorithms.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Document	
Period Covered	M6-20
Deliverable No.	D2.2
Deliverable Title	Report on Data Placement and Migration Methodologies
Editor(s)	Feroz Zahid
Author(s)	Kyriakos Kritikos, Feroz Zahid, Somnath Mazumdar, Daniel Seybold, Yiannis Verginadis
Reviewer(s)	Daniel Baur, Paweł Gora
Work Package No.	2
Work Package Title	Architecture and Data Management
Lead Beneficiary	Simula Research Laboratory
Distribution	PU
Version	1.0
Draft/Final	Final
Number of Pages	57

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

List of Figures

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1 Introduction

Big Data is one of the major current trends in ICT. In areas as diverse as social media, business intelligence, information security, Internet of Things (IoT), and scientific research, a tremendous amount of both structured and unstructured data is created and collected at a speed surpassing what we can handle using traditional data management techniques. Users create content, behaviour is recorded, sensor data are collected, and experiments are run, to mention just a few potential producers and sources of big data. Within the large amount of data produced, a great potential resides in the form of undiscovered values, structures, and relations. To facilitate realising this potential, which is turning out to be a new competitive advantage to businesses and to researchers [1], enterprises are increasingly relying on computational and data storage capacities offered by the Cloud. In the Melodic project, we are developing a middleware platform that enables data-intensive applications to run within defined security, cost, and performance boundaries seamlessly on geographically distributed and federated Cloud infrastructures. Melodic thereby realises the potential of heterogeneous Cloud environments by transparently taking advantage of distinct characteristics of available private and public Clouds.

An important challenge in *Cross-Cloud* application deployments is *data-awareness*, which refers to the deployments that consider locations of the data sources into account. Applications consume and process data, potentially originating from various data sources, as well as generate data to be stored at various on-Cloud and off-Cloud locations. The lack of data-awareness may lead to poor application performance and costly data migrations. For instance, when application deployment decisions do not take in account locations of the existing data sources and their size, as well as any security and privacy constraints applicable, application performance degrades and/or expensive data migrations incur. In addition, intelligent and pre-emptive data placement and migration strategies are important for efficient data-intensive computing in Cross-Clouds. For instance, unlike applications, data placement in Clouds is generally subjected to long-term storage selection as migration generally incurs high costs. Moreover, the initial Cloud selection can also potentially affect the subsequent placement of applications due to *data gravity*. Data gravity is an analogy of the nature of data and its ability to attract applications and services. As the size of data grows, services and applications are more likely to be placed near the data, rather than vice versa [2].

Trust has also remained a major issue hindering the broader adaptation of Cloud services by enterprises for data-intensive computing [3], [4]. It is a common perception that, due to lack of control and transparency, data stored in the Cloud is prone to theft, misuse and unauthorised access. Hence, it is critically important that the user-defined data constraints and requirements are satisfied during the complete application and data life-cycle.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

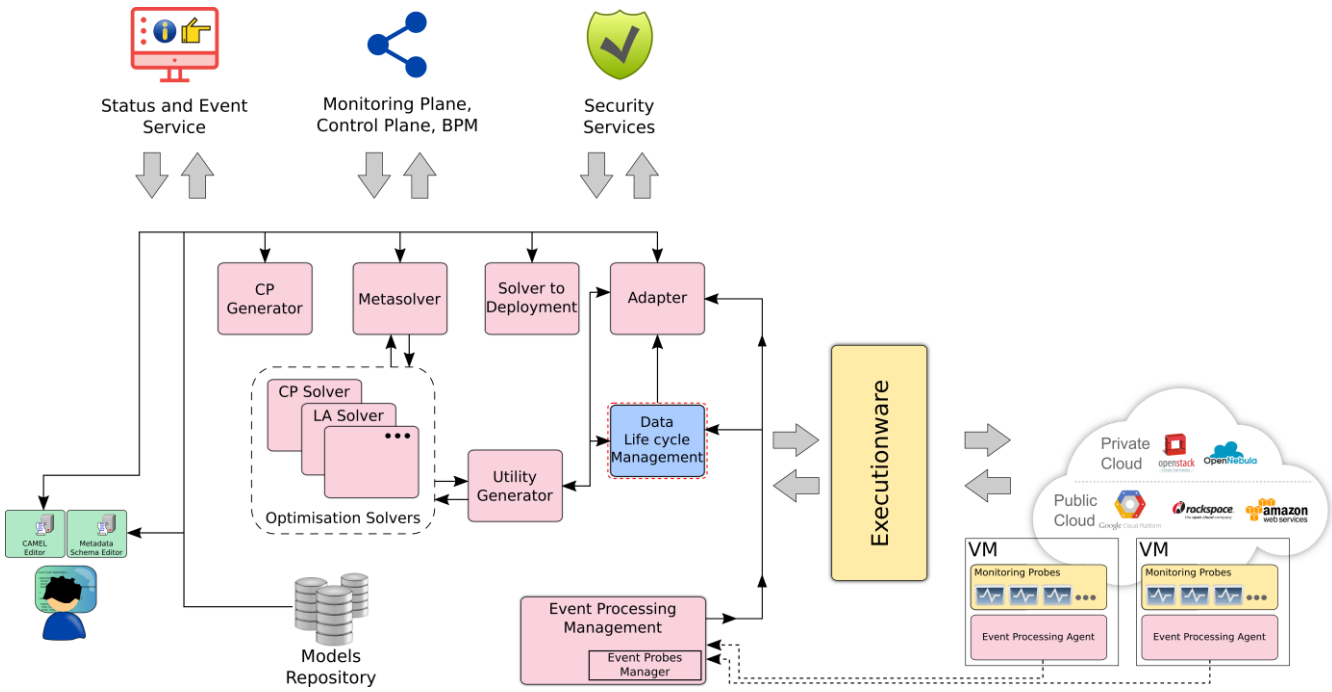


Figure 1: Overview of the Melodic Upperware

The Melodic *Upperware* includes a dedicated component, Data Life-cycle Management System (DLMS), which enables data-aware application deployment and optimisation in Cross-Cloud environments. In this deliverable, we report on research and development activities associated with designing data placement and migration methodologies for Cross-Cloud data management through the DLMS. The main functionality that DLMS offers includes:

- Management of data sources on behalf of the Melodic user
- Optimal data placement in the Cloud based on user-defined data placement requirements, constraints, and associated costs
- Keeping user-defined data requirements satisfied throughout the data lifetime
- Assignment of a *utility value* to the proposed deployment solutions with respect to current and proposed data source and application component locations

As shown in Figure 1, the DLMS interacts with the *Utility Generator* and the *Adapter* components of the Upperware. For each solution proposed by the Melodic Optimisation *Solvers*, the Utility Generator consults DLMS, which assigns a utility value based on the penalty for adopting the proposed configuration taking into account the effect the proposed solution will have on data sources and their access by the application components. To elaborate further, the utility value represents the degree to which a solution is favourable by the DLMS, considering any data migrations required and impact on the application performance by the prescribed placement of application components and data sources. The DLMS computes utility value based on historical data access patterns reflecting affinities between application components and data sources,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

dataset characteristics, average network latencies and throughput between data centres, Cloud provider costs, and predictions from past DLMS decisions, as implemented through the DLMS algorithms.

The utility value from the DLMS will then be used by the Utility Generator to assign an application-level utility value to the candidate solution as per the defined utility function. In autonomic computing utility functions have been extensively used to express the goodness of a particular application configuration as seen and perceived by the application owner [5]. Further, the Adapter, which is the component responsible for analysing and validating a new deployment model and defining a number of reconfiguration action tasks to be executed in a specific order, consults DLMS for specific data migrations and configurations needed for a deployment reconfiguration. The detailed architecture of the Upperware is discussed in [6].

1.1 Scope of the Document

This document is intended for the general audience interested in learning about the state-of-the-art data management techniques in the Clouds, and the details about the DLMS design and methodologies in the Melodic platform. Parts of the document require a high-level understanding of the Cloud technologies and the Melodic platform, for which readers are referred to [7] and [6]. The document is also complemented by the Melodic deliverable *D3.2 Business logic for supporting the complete data and data-intensive application life-cycle management* [8], which presents implementation details of the DLMS.

1.2 Structure of the Document

The rest of this document is logically divided into two parts. In the first part, comprising Chapter 2 through Chapter 5, a state-of-the-art analysis of data management in the Cloud is provided. In Chapter 2, data storage technologies are discussed and a framework with a taxonomy and non-functional requirements is provided. Chapter 3 provides a review of database management systems, while file systems are reviewed in Chapter 4. We present a comprehensive survey of the existing data placement and migration methodologies in the Cloud in Chapter 5. The second part of this document reports on the research and design activities leading to the Melodic DLMS component. Chapter 6 details the data and meta-data schema used for data modelling in Melodic. The architecture of the DLMS is discussed in Chapter 7. Chapter 8 provides DLMS algorithms used for optimising data placement and migrations in Cross-Clouds. We conclude in Chapter 9 while also supplying some future work directions.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

2 Data Storage Technologies

Data storage technologies refer to the tools, techniques, and methods for storage, management, and access of datasets. In the context of big data in the Cloud, data management has become an important issue, challenged by the rapidly growing scale of the problem. The amount of digital data in our world has become enormous and is growing at exponential rates. It is estimated that the size of the *digital universe* will grow from 4.4 zettabytes in 2013 to 44 zettabytes by 2020 [9]. A major factor influencing this rapid data explosion is the growing popularity of data-intensive applications in a variety of domains. Formally, big data refers to the datasets whose size is beyond the ability of conventional software tools to store, manage and analyse. However, size or *volume* is not the only feature that prescribes which data should be classified as big data and which should be not. Besides plausibly the high volumes of data, big data also imposes challenges in handling the v*ariety* of these volumes with different forms (structured and unstructured), at a considerably high *velocity* or transfer rate. The volume, velocity, and variety, together make up the three most important challenges associated with the management of big data in communication networks, and are referred to as the three Vs in the literature [10]. Besides these three Vs, it is equally important that the *value* of big data can be extracted, even in the presence of *veracity* or uncertainties in them [11]. An important point to note here is the fact that as more and more digital data is being produced in different areas, many of the computational problems formerly known to be associated with structured or low volume data, for instance data query, are converging to big data problems -- pushing the need for efficient data management and processing [12], [13].

In this chapter, we present a taxonomy of storage systems relevant for the big data management in distributed systems. Moreover, we also provide a set of non-functional requirements that such data storage systems should satisfy to efficiently support data-intensive applications in the Cloud.

2.1 Taxonomy

We have developed a reference taxonomy for the classification of data storage technologies relevant to the Cloud environments. As shown in Figure 2, we classify data storage into two main categories: Database Management Systems (DBMSs), and File Systems. The taxonomy presented in each of these respective categories is based on the classification presented in the literature for file systems [12], [13] and DBMSs [14], [15]. The DBMS-based storage systems can be divided into

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

two types based on the *storage model* used: Relational storage and Non-Relational storage, each of which is further classified into sub-categories. An overview of the DBMSs is provided in Chapter 3. File systems can be basically classified into local and distributed file systems (DFSs). The DFSs are classified based on the architectural model used in the distribution. Both Client/Server and Clustered Distributed architecture models are in common use. An overview of the DFSs is provided in Chapter 4.

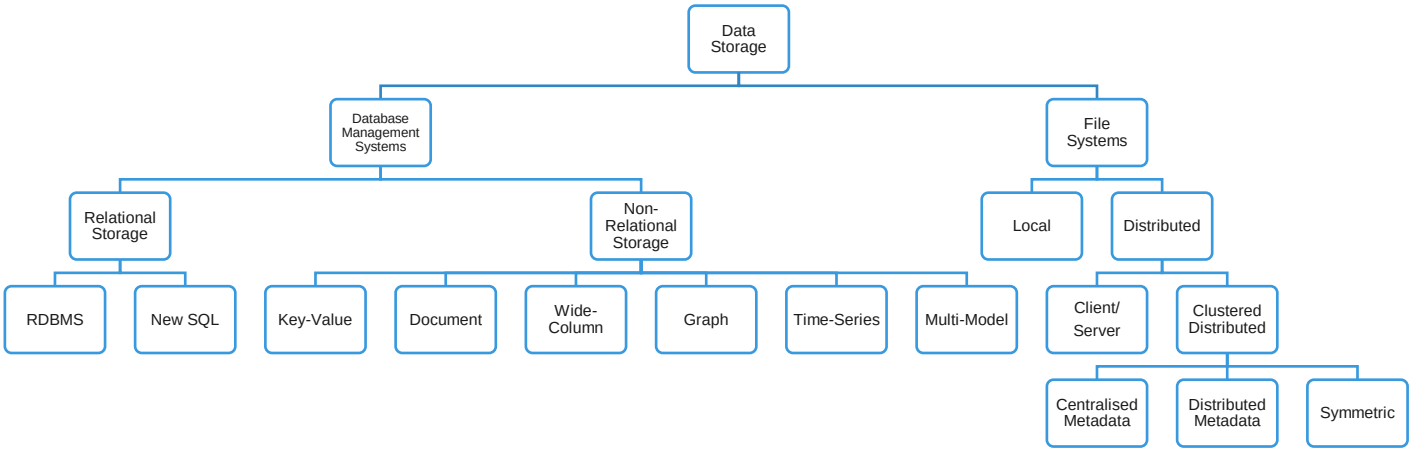


Figure 2: A taxonomy of the data storage technologies

2.2 Non-Functional Requirements

The primary non-functional goals of a traditional distributed storage system are increased availability, better scalability, higher performance, better workload distribution among the storage nodes and higher fault tolerance. However, big data management in the Cloud raises the level of complexity by adding more unique management features such as: on-demand resource access (related to elasticity), heterogeneous hardware or storage model support and faster data retrieval through better data query interfaces, and security. In Cross-Cloud environments with heterogeneity of data sources, as targeted by Melodic, a unified namespace and transparent access for the applications is also important. The aforementioned challenges have led us to define a set of non-functional requirements that existing DBMSs and DFSs should satisfy. The requirements are listed in Table 1.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 1: Non-functional requirements for data storage systems

Requirements
High Performance Typical performance metrics are read/write throughput and latency. Throughput refers to the number of requests that can be processed per unit time, while latency defines the time required to transfer a request to the client and the storage system. Performance benchmarking suites for both distributed file systems [10] and DBMSs [11], [12] have been presented in the literature.
Scalability Scalability is the ability of a system to sustain increasing workloads by making use of additional resources [14]. Scalability can be defined by the terms *scale-up*, *scale-down*, *scale-out* and *scale-in* in order to manage growing/shrinking workloads. Scale up/scale down or vertical scaling applies by adding (or reducing) more computing resources to a single node, while horizontal scaling, namely scale-out (i.e., adding nodes to a cluster) and scale-in (i.e., removing nodes from a cluster) applies for a distributed data storage system comprising multiple nodes [15]. The scalability of a DBMS is measured by its ability to exhibit constant latency and a proportionally growing throughput with respect to the number of nodes and the workload size for horizontal scalability or the applied computing resources for vertical scalability [16]. DFSs based on a centralised architecture depend on a single server bottleneck which limits their scalability. However, techniques such as *multi-threading* and *caching* are commonly employed to improve scalability [17]. Many performance benchmarking suites also support scalability analysis [10] [11], [12].
Consistency Traditionally, database transactions are needed to conform to the ACID guarantees, comprising *Atomicity*, *Consistency*, *Isolation*, and *Durability*. However, maintaining ACID transactional guarantees over large distributed infrastructures have been proven to be hard [18]. For instance, when data is replicated over geographically-dispersed locations, maintaining consistency can have a substantial negative effect on the availability of the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

system [19], [20]. Alternative guarantee approaches, such as BASE (*Basically Available, Soft State*, *Eventual consistency*), are practically sufficient for most Cloud applications [21], they make Cloud transition harder for some enterprise applications that heavily rely on transactions. Supporting transactional databases in Cross-Clouds is subject to challenges physically bounded by latency between distributed Cloud data centres.

In DFSs, *synchronisation* of the data copies across multiple replicas is important. Both *synchronous* and *asynchronous* methods are used. In synchronous methods, the request to a modified data block is blocked until all copies are updated, which slows the query. While, in asynchronous methods, requests accessing out-of-date copy of data are allowed [17]. More advanced *semi-asynchronous* methods have also been developed to bridge the trade-off between consistency and performance.

Cache consistency is also a problem of synchronisation of data which is stored in cache. Several methods, such as Write Only Ready Many (WORM), *Transactional locking*, and *leasing* are commonly employed for achieving cache consistency in both DFSs and DBMSs.

Elasticity

Elasticity is tightly coupled to horizontal scalability and enables the overcoming of sudden workload fluctuations by dynamically scaling the cluster without any downtime [16].

Shared-Nothing architectures are particularly useful for achieving rapid elasticity in Cloud computing systems. In a shared-nothing architecture, each node in a distributed system is independent and self-sufficient. Shared-nothing architectures are employed in both DBMSs and DFSs.

Fault Tolerance

Fault-tolerance refers to the ability of the system to continue operating in case of failures. Replication is a common way to achieve fault-tolerance. This includes, among others, continuous incremental backup of data, duplicate transactions logs for warm-failover, and periodic check-pointing.

Transparency

Transparency refers to the property of a storage system that the users have access to the same file system structure / storage model regardless of their location, and irrespective of the underlying system implementation.

Unified namespace is a common way to achieve transparency. Further, in Cross-Cloud environments where applications access data sources, for example from multiple DFSs and

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

storage systems, a global unified namespace can warrant easy access and manipulation of data.
Application Access Application access refers to the requirement of easy data access interface. Many DBMSs provide easy access and manipulation of data stored using the Standard Query Language (SQL). DFSs clients are usually provided through programming language libraries, command line interfaces, and REST APIs.
Security and Privacy An important requirement for the data storage systems is their ability to provide mechanisms to ensure security and privacy of the data stored. This includes support for authentication, authorisation, confidentiality, integrity, and auditability.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

3 Database Management Systems (DBMS)

In the last decade, the landscape of database management systems has grown significantly. Enabled by the Cloud and driven by new application domains such as big data, distributed DBMSs (DDBMSs) with heterogeneous storage models moved into the focus of academia and industry [22], [23]. DDBMSs are built on a shared-nothing architecture and promise to cater for typical big data related requirements, such as scalability, elasticity, and availability by running on commodity hardware or in the Cloud [24]. In this context a diverse set of new data storage models appeared on the DBMS landscape besides the relational storage model, which has been the common storage model for DBMSs over the last several decades.

An up-to-date overview of the current data storage models of DBMS is provided in Figure 2, which is distilled from current trends in the DBMS, cloud and Big Data research communities. The data storage models are classified into the two main categories, relational and non-relational storage. Relational storage offers the common SQL query interface and ensures ACID consistency [25], as well as referential and functional integrity. Non-relational storage (commonly referred as NoSQL) provides more heterogeneous storage models with weaker consistency guarantees that are commonly referred to as BASE consistencies [26]. Yet, the non-relational storage models ease a distributed architecture and favour horizontal scalability, elasticity and high-availability of the respective DBMS [27].

In the following, an overview of the common relational and non-relational storage models of Cloud and big data related DBMSs is provided. In addition, common DBMS representatives[1] of each data storage model are briefly presented.

3.1 Relational Storage

Relational storage has been the common data storage model for many decades, represented by a relational DBMS (RDBMS). Yet, driven by the big data evolvement, also so called NewSQL DBMS appeared in the DBMS landscape besides the RDBMS as depicted in Figure 2. NewSQL adopt the relational storage model with the focus on a distributed architecture of the DBMS [28]. In the following, both RDBMS and NewSQL are introduced.

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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

3.1.1 Relational DBMS (RDBMS)

A Relational DBMS (RDBMS) stores data as tuples, forming an ordered set of attributes. A relation consists of sets of tuples such that a tuple represents a row, an attribute is a column and a relation forms a table. Tables are defined using a static, normalised data schema and different tables can be referenced using relationship. RDBMSs provide SQL as the established interface for generic data definition, manipulation and querying. Moreover, a rich support for analytical queries and aggregations is generally available. Due to the relational data model and the strong ACID consistency guarantees [25], RDBMSs offer only limited horizontal scalability or elasticity. Popular RDBMSs include MySQL[2] and Postgres[3] in the open source context and IBM DB2[4] and Oracle RDBMS[5] in the enterprise context. In addition, RDBMSs are also offered as Database-as-a-Service (DaaS) by major public Cloud providers, such as Amazon RDS[6] or Azure SQL[7].

3.1.2 NewSQL

As the traditional relational data model only provides limited support for data partitioning, and thus, limited horizontal scalability and elasticity capabilities, NewSQL DBMSs try to bridge this gap. Initiated by Google Spanner [29] in 2013, NewSQL DBMSs are built upon the relational data model and SQL as the standardized interface but target horizontal scalability and elasticity to enable large-scale distribution [22]. However, only a few NewSQL DBMS are built using novel architectures, whereas many NewSQL DBMSs are built upon existing DBMS solutions [28]. The NewSQL landscape is rather new and currently evolving, but some adopted representatives are VoltDB[8], CockroachDB[9] and NuoDB[10].

[2] https://www.mysql.com
[3] https://www.postgresql.org/
[4] https://www.ibm.com/analytics/us/en/db2/
[5] https://www.oracle.com/database/technologies/index.html
[6] https://aws.amazon.com/rds
[7] https://azure.microsoft.com/services/sql-database/
[8] https://www.voltdb.com/
[9] https://www.cockroachlabs.com/
[10] https://www.nuodb.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

3.2 Non-Relational Storage

Non-relational storage or NoSQL has significantly evolved over the last decade. Even though early non-relational databases were available as early as 1970s [30], a large-scale popularity started with Amazon Dynamo in 2007 [31] and Google Bigtable in 2008 [32]. Since them, the non-relational DBMS landscape has constantly evolved with respect to the data storage models and existing DBMSs. In the following the most common non-relational storage models with respect to big data are presented, which are depicted as the respective Non-Relational Storage sub-categories in Figure 2. In addition, for each storage model, we give examples of the most common representative DBMSs.

3.2.1 Key-Value

The key-value storage relates to 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 create, read, update, delete (CRUD) operations of items referenced by their key. Inspired by the consistent hashing approach of Dynamo [31], key-value DBMS are optimised for operation in large-scale clusters and support horizontal scalability and elasticity. Popular key-value DBMSs are Riak[11], Redis[12] and the Amazon's DBaaS offering S3[13].

3.2.2 Document

The document storage model builds upon a similar data model as in key-value storage. Yet, in contrast it defines a structure on the values in formats, such as XML or JSON. These values are referred to as documents, but usually without fixed schema definitions. Compared to key-value DBMSs, they allow for more complex queries, as document properties can be used for indexing and querying. Document storage models also support advanced queries for aggregations and analytical functions. Further, document DBMSs support a distributed operation and enable

[11] http://basho.com/products/
[12] https://redis.io/
[13] https://aws.amazon.com/s3/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

horizontal scalability and elasticity even for geographically distributed clusters. Common document storage DBMSs are MongoDB[14], Couchbase[15] and the DBaaS offering Azure Cosmos DB[16].

3.2.3 Wide-Column

Wide-column storage is inspired by Google's Bigtable [32] and Apache Cassandra [33]. Wide-column storage stores data by columns rather than by rows. This enables storing large amounts of data in bulk and for efficiently querying over very large, structured data sets. A column-oriented data model does not rely on a fixed schema. Instead, it provides *nestable*, map-like structures for data items which improves flexibility over fixed schemas. Column-oriented DBMS support horizontal scalability and elasticity for large scale and geographically distributed clusters. Besides Google's DBaaS offering Bigtable[17], Apache Cassandra and Apache HBase[18] are popular wide-column DBMS.

3.2.4 Graph

Graph-based DBMS are inspired by graph theory. They use graph structures for data modelling, and nodes and edges represent (and store) data. 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. Due to their graph-focused data model which implies strong relations between data entries, graph storage only offers limited support for horizontal scalability and elasticity. Common graph storage DBMS are Neo4j[19] and TitanDB[20].

3.2.5 Time-series

Time-series (TS) storage is driven by the need for monitoring of large-scale Cloud and IoT applications and infrastructures, which require horizontally scalable DBMSs with analytical query support. Therefore, time-series DBMS typically built upon existing DBMS data models, preferably

[14] https://www.mongodb.com/
[15] https://www.couchbase.com/de
[16] https://azure.microsoft.com/services/cosmos-db/
[17] https://cloud.google.com/bigtable/
[18] https://hbase.apache.org/
[19] https://neo4j.com/
[20] https://github.com/thinkaurelius/titan

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

key-value or column-oriented, and add a dedicated time-series data model on top [34]. The TS storage model is built upon data points which comprise a timestamp, an associated numeric value and an unstructured set of metadata. TS DBMS are typically optimised for write performance and they support horizontal scalability and elasticity for distribution the load across large scale clusters. In addition, TS DBMS often provide additional tools for the collection and visualization of the monitoring data. Common TS DBMS are InfluxDB[21], Prometheus[22] and Axibase[23].

3.2.6 Multi-model

Multi-model storage combines different storage models into a single DBMS to improve flexibility, e.g. providing document storage and graph storage tables to cater for scalability or improved query support, depending on the respective table data. Common multi-model DBMS are ArangoDB[24] and OrientDB[25].

[21] https://www.influxdata.com/
[22] https://prometheus.io/
[23] https://axibase.com/
[24] https://www.arangodb.com/
[25] https://orientdb.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4 Distributed File Systems (DFSs)

A DFS provides a permanent storage to the data objects through a set of storage resources connected by a communication network [35]. DFSs allow multiple distributed nodes to share files, which is a common requirement for many distributed applications. The data is created and consumed by the client applications using a unified DFS fabric. The DFSs offer scalability, fault tolerance, concurrent access of the files, and a unified namespace.

As shown in our taxonomy of Figure 2, the DFSs can be classified into two broad classes based on the architecture model used: Client/Server and Clustered Distributed DFSs. In the following, we discuss each of these classes and provide examples of the representative DFSs.

4.1 Client / Server Model

In a Client/Server architecture model of DFSs, the data storage server provides a standardised view of its local file system to the clients on a network through a communication protocol. The Client/Server DFS model was used by early network file systems, such as Sun Microsystem's *Network File System*, and is still popular and has also found its way into Cloud computing systems. An example of a Cloud-supported Client/Server based DFS is GlusterFS[26].

GlusterFS (combined Gnu and cluster FS) is an open source, POSIX compatible, DFS that aggregates disk storage resources from multiple servers into a single global namespace. GlusterFS relies on an elastic hashing algorithm instead of a distributed metadata model. The built-in scalability feature can support several petabytes while supporting thousands of clients running on the commodity hardware. It also can support multiple workloads using various local file systems that supports extended attributes and also provides (geo)-replication, quotas, and snapshots. The master nodes are deployed as storage bricks which expose a local file system to the network via a GlusterFS daemon. A custom protocol over TCP/IP, Infiniband or sockets direct protocol is employed for the communication between the client and the servers.

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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4.2 Clustered Distributed Model

Clustered Distributed architecture model of DFSs is the most commonly employed architecture model in modern DFSs. In a clustered distributed model, metadata and data are decoupled and stored in a set of distributed nodes in a cluster. Clustered distributed model can be further classified into three models, based on how the metadata is stored, as explained in the following.

4.2.1 Centralised Metadata

In clustered distributed DFSs which are based on centralised metadata, the metadata is stored in a centralised metadata node, while other nodes store the data. Hadoop Distributed File System (HDFS)[27], Integrated Rule-Oriented Data System (iRODS)[28], and MooseFS[29] are some examples of such DFSs.

Hadoop Distributed File System is an open source, DFS implementation of Google's file system (GFS) that can run on commodity hardware. HDFS consists of a *NameNode* (the master) and multiple *DataNodes* (the slaves). The master node manages the file system namespace and controls the file access by clients. It also determines the mappings of blocks to DataNodes which are also responsible for serving read and write requests from the file system's clients. HDFS divides files into same sized data blocks for allocating them to DataNodes. DataNodes are responsible for storing data blocks.

Integrated Rule-Oriented Data System (iRODS) is a policy-based, distributed data management system. The most important feature of iRODS is its rule engine. It allows data to be managed with policies and expressed as actionable objects which can further be executed using microservices. Rules can be invoked by users using a command line or API. Apart from the engine, iRODS supports complex metadata. The complex metadata can be defined by the users using the triplet (key, value, unit) format provided by the iRODS. This feature provides more control over the metadata customisation.

MooseFS is another open source, POSIX compliant, fault-tolerant, centralised metadata based clustered DFS. It distributes the user data across a large number of servers, which are perceived by the user as a single virtual disk.

[27] http://hadoop.apache.org/
[28] https://irods.org/
[29] https://moosefs.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4.2.2 Distributed Metadata

In clustered distributed DFSs which are based on distributed metadata, the metadata together with the data, is distributed among the participating nodes in the DFS cluster. CephFS[30] is a popular example of such DFSs.

CephFS is an open source, distributed, a POSIX-compliant file system that uses a Ceph Storage Cluster to store its data. Ceph is designed to offer block, object, and file storage to improve performance, reliability, and fault tolerance by data replication. A Ceph Storage Cluster supports multiple metadata servers. Each Ceph metadata server stores the metadata on behalf of the CephFS. A specialised replication algorithm, named as CRUSH (Controlled Replication Under Scalable Hashing) enables the Ceph Storage Cluster to scale, rebalance, and dynamically recover from the failures.

4.3 Symmetric

Symmetric also called as peer-to-peer (P2P) file systems employ P2P technologies [36] as the basis of their architecture. Both the metadata and the data is distributed among a set of nodes, and each node offers an equivalent functionality [37]. Examples include Red Hat GFS, PVFS [38], and ivy [39].

Red Hat GFS is a POSIX compatible cluster file system which offers a single, consistent view of the file-system namespace across the GFS nodes in the cluster. The nodes in GFS operate in share-nothing architecture, each having the same set of roles.

Parallel Virtual File System (PVFS) is a parallel file system that aims at offering high bandwidth concurrent read/write operations for parallel applications. The PVFS enables user-controlled partitioning of data across disks on the participating I/O nodes. Ivy supports concurrent read-write file system access using a P2P system. The whole file system is built around a set of logs, one log from each participating node. Each node writes its own log but reads logs from all other nodes. The logs are stored in a distributed hash (DHash) table. Ivy offers both conventional file system interface as well as traditional network file system (NFS) like semantics.

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

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

5 Data Placement and Migration Methodologies

In this chapter, we provide a state-of-the-art analysis of the existing data placement and migration methodologies, algorithms, and tools.

5.1 State-of-the-Art Analysis

The data placement and migration methodologies relate to both the initial placement of data in the Cloud and the subsequent migrations of the data based on an optimisation criteria. The data placement solutions, in general, also result in data migration decisions when considering the current locations of data sources into account.

The data placement problem in Clouds can be formulated as a variance of the knapsack packing problem, which is NP-Hard [40]. The main aim is to find a solution with optimal mapping of datasets to respective resources (VMs in Cloud datacentres). To solve this problem, different optimisation objectives can be used. The most common criteria relate to the minimisation of the amount of data transfer needed during the placement, and data storage costs. However, other criteria representing the impact of the data source locations on application performance, for instance, can also be utilised.

In [41] (bdap), a big data placement algorithm is proposed which goes to the level of VMs. This algorithm does take into account the existence of big datasets as well as of intermediate data produced during the application runs. It relies on using a meta-heuristic approach for the solution by applying a genetic optimisation technique over a data interdependency matrix.

The approach in [42] (Xu et al.) focuses on a coarser level of granularity to solve the data placement optimisation problem. It tries to optimise the overall number of data transfer schedules by applying genetic programming to the respective optimisation model produced which does consider data centre capacity constraints and the non-replication of datasets.

Yuan et al. [40] propose a k-means dataset clustering algorithm that can be executed at two time points: (a) at application build time; (b) at application execution time. At application build-time, the algorithm constructs a data dependency matrix which is then transformed into another form, where similar items are paired together by applying the Bond Energy Algorithm (BEA) [43] and finally the items are clustered based on their dependencies/similarities by following a recursive binary partitioning algorithm. An interesting aspect of the latter algorithm is that it does take into account the fact that data can grow over time and considers the use of bounds to restrain the further use of a datacentre when data size goes above them. During application runtime, the data placement is executed mainly over new datasets which are positioned accordingly in the most

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

suitable data centres by following a similar approach. However, in this case, the existing placement can be invalidated due to different reasons: (a) there is no further space to devote to a new dataset in the datacentre selected; (b) new workflows enter the system. To this end, a restricted form of the application build-time algorithm is applied to find a new solution which is applied incrementally from the current data placement solution. Please note that the proposed approach goes beyond data placement to also deal with the deletion of obsolete data. Such data unnecessarily occupies storage, which could be freed, and lead to an increase in the storage cost.

Kaya et al. [44] propose an approach which is able to simultaneously decide about the optimal data placement and task scheduling. To achieve this, they model the workflow as a hypergraph and then extend the PaToH hypergraph partitioning algorithm [45] to employ heuristics that attempt to simultaneously reduce the computational and storage load while trying to minimise the total amount of file transfers. The proposed approach is applied over the granularity of data centres. However, it seems that currently it cannot be dynamically applied at runtime.

Yu and Pan [46] advocate that traditional placement algorithms incur a high cost on storing and transferring of logs as well as an increased runtime. To this end, they propose the use of sketches of request traffic in the distributed infrastructure to lower the data placement overhead. Such sketches represent data structures that approximate properties of a data stream via a sublinear space. In the proposed work, two types of sketches are maintained in a sliding window fashion: (a) sampling sketches covering the uniform event sampling in the stream and (b) counting sketches covering the frequency of event occurrence in the stream. These two sketches types are exploited to construct a hypergraph *sparsifier*, via employing a randomised heuristic and an interactive protocol between the sketches and the data processing controller, on which a hypergraph partitioning algorithm is applied to derive the respective data placement decisions.

Zhang et al. [47] propose a Mixed-Integer Programming (MIP) model for modelling the big data placement problem which takes into account both the data access cost as well as the storage limitations in the data centres considered. This model is then solved through the use of a *Langrangian* relaxation-based heuristic algorithm.

Lan et al. [48] focus on optimal variable big data stream partitioning, especially for the IoT data. To this end, they propose a clustering-based particle swarm optimisation (PSO) search method which relies on statistical feature extraction for stream classification. Prior to training the classification model, redundant features can be discarded by applying conventional feature selection techniques like correlation feature selection, information gain, PSO and genetic algorithm. In the classification phase, the basic learner of Hoeffding Tree is exploited.

Kayyoor et al. [49] focus on minimising the query span for query workloads through applying replica selection and data placement algorithms. Their algorithms can be applied on both multi-site data warehouses and general-purpose data centres. They also assume that a query workload

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

trace always provides the set of data that need to be accessed. Query workloads are represented as hypergraphs mapping the data as nodes which map to hyperedges over the nodes. The main goal is to optimally map the nodes to a subset of all clusters by also taking into account storage capability constraints on the clusters such that the average cost among all queries is minimised. This goal is achieved by considering a hypergraph partitioning algorithm (HPA) as a blackbox, checking and selecting different partitioning algorithm classes for this blackbox and then exploiting data replication in order to minimize the average query span. The implementations considered include: (a) Iterative HPA: the HPA algorithm is iteratively executed until no space is left, (b) dense sub-graph-based: a dense sub-graph algorithm is used to exploit the redundancy, (c) pre-replication: identification of a set of nodes for replication, modification of the input graph and application of the HPA algorithm for finding the most optimal data placement and (d) local move-based: opposite to the previous case, a certain solution from HPA is used as a baseline for replicating a small set of data items at a time. A proposal for extending the four algorithm classes for supporting three-way replication is also discussed.

Volley [50] analyses logs of the data centre requests using an iterative optimisation algorithm based on data access patterns and client locations, and recommends data migrations to the cloud service to increase performance.

Scalia [51] is a cloud brokerage solution that continuously adapts the placement of data based on its access patterns and subject to the optimisation objectives, such as storage costs.

The data placement and migration algorithms discussed above are generalised and can be applied to both DFSs and DBMSs storage technologies. Some solutions, however, are more specific to specific DFS and DBMS technologies, as discussed in the following.

Hsu et al. [52] propose an extension of Hadoop over heterogeneous environments focusing on achieving: (a) dynamic data redistribution at each data node before the Map phase by considering the data processing speed of each VM calculated through profiling, (b) VM mapping for reducers, a feature not supported by Hadoop, based on partition size and the availability of VMs on the respective physical machines, as well as (c) optimal reducer selection through assigning reducers with higher workloads to VMs with higher processing speed.

A data-group-aware placement scheme is proposed in [53] (daware) for Hadoop. This scheme exploits data access patterns from data access logs and then extracts optimal data groupings and reorganises the data allocation in order to achieve maximum parallelism per group to better balance the respective load. The scheme operates at the level of the rack/cluster. It also recursively employs the BEA algorithm in order to transform the original data dependency matrix into a clustered one. To also take into account the vertical relationships between data apart from the horizontal ones so as to ensure that the blocks on the same node have a reduced probability to be part of the same group, the Optimal Data Placement Algorithm (ODPA) is applied over a sub-

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

matrix obtained from clustered dependency matrix, thus considering the dependencies between the data already placed and those being placed.

SWORD [54] proposed a workload-aware data partitioning and placement approach for online transaction processing (OLTP) workloads. The implementation is based on PostgreSQL RDBMS.

A data management middleware targeting NoSQL databases has been presented in [55], which makes abstraction of multiple Cloud storage technologies, and follows a policy-driven approach for making data placement decisions.

5.2 State-of-the-Art Comparison

In order to evaluate how suitable are the data placement and migration methodologies performed by the state-of-the-art algorithms selected from the literature, we have compiled a set of criteria. Please note that we use these criteria not to analyse only pure data placement algorithms but also data placement strategies in the sense that such strategies also include data placement algorithms that need to be also evaluated.

- Fixed data positioning: Due to regulations or privacy reasons, specific datasets of an organisation might need to reside in a data centre or in a specific country. We consider whether the data placement strategies and algorithms are capable of including fixed data positioning.
- Constraint solving technique: Which constraint solving techniques are employed.
- Host Granularity: This criterion indicates what the granularity of the placement of big data is. A coarse-grained granularity can means the selection of the data centre on which a dataset can be placed. A fine-grained granularity means the selection of the VMs where data needs to be placed.
- Intermediate data handling: while applications are running, they produce intermediate data which are most of the times consumed by other application tasks. The production and use of such data leads to the case where the current data placement decisions need to be invalidated in order to immediately grab better optimisation opportunities.
- Multiple application handling: An application might have multiple instances running which could operate over the same datasets while a system might need to support the placement of multiple applications that share a common set of datasets.
- Data size uncertainty: Data might be processed by applications but they are not always static in nature. In many cases, they can grow in even unanticipated paces. However, this data size increase, even if it is expected, can do lead to invalidating data placement decisions.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- **Replication:** Whether placement decisions consider data replication.
- **Optimisation Criteria:** While data transfer cost plays an important role in big data placement, it is not the sole factor that needs to be optimised. In particular, for the Cross-Cloud data placement, it is important to consider other criteria that affect the performance of the application.

A summary of our comparison between selected state-of-the-art approaches is given in Table 2.

Table 2: Comparison of the State-of-the-Art data placement approaches

Approach	Fixed DS	Solving Technique	Host Gran	Int. data	Multi Apps	Data Size	Rep	Criteria
bdap [41]	Yes	Meta-heuristic (genetic programming)	Fine	Yes	No	No	No	Communication Cost
Xu [42]	No	Genetic programming	Coarse	No	No	No	No	Data transfer amount
Yuan et al. [40]	Yes	Recursive binary partitioning (BEA)	Coarse	Yes	Yes	Yes	No	Data transfer amount
Kaya et al. [44]	No	Hypergraph partitioning	Coarse	No	No	No	No	Data transfer amount
daware [53]	No	Recursive Clustering via BEA and ODPA	Fine	No	No	No	No	Data transfer amount
Yu and Pan [46]	No	N-way Hypergraph partitioning through sparsification	Fine	No	No	No	No	Cut weight calculated based on edge weight and partition number

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Zhang et al. [47]	No	Lagrangian relaxation of MIP	Coarse	No	No	No	No	Data access cost
Hsu et al. [52]	No	-	Fine	No	No	No	No	Processing speed
Lan et al. [48]	No	Clustering-based particle swarm optimisation search	Fine	No	No	No	No	Volatility, Autoregressive Moving Average, Hurst component, distance
Kayyoor et al. [49]	No	Hypergraph partitioning - different classes of algorithms	Coarse	No	No	No	Yes	Average query span

As we can see from Table 2, there are shortcomings in the state-of-the-art data placement and migration techniques. First, we can deduce that very few approaches consider the existence of fixed data sets that cannot be moved. Out of these approaches, the policy enforcement ones are discerned based on the fact that policy enforcement could be highlighted as the usual way to deal with such datasets by considering that policies actually prescribe the data gravity. However, we need to stress here that copying/enforcing policies is not just an act of denying or accepting a data placement request. Instead, policies should be seen as global or data-specific constraints that need to be taken into account when attempting to solve the big data placement problem.

Second, most of the approaches in data placement focus mainly on the initial positioning of data and do not interfere with the actual runtime of the respective applications. However, most of the application classes in the real-world are really dynamic in nature, can have different variation points in load and certainly produce additional data which need to be promoted accordingly to the next computation steps. As such, data placement cannot be just a one-short process but a continuous one which is continuously run in order to re-evaluate placement decisions as well as reach new ones, when the respective need arises.

Third, concerning the optimisation criteria considered, it seems that half of the approaches focus on just one criterion which is related to the minimisation of the data transfer amount, and not the actual data transfer costs which varies from Cloud to Cloud. Moreover, the impact on the application performance based on the location of data sources is not considered in several

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

approaches. This could enable, for instance, recommendation of moving a data set to another data centre when it could be predicted (from the historical records) that the respective data transfer costs would be significantly less considering the performance gain during the application life-time In general, we believe that optimisations based in a single objective are not sufficient for a Cross-Cloud deployment solution, targeted by the Melodic DLMS.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

6 Data Modelling

In Melodic, before application and data placement decisions are made, both the data as well as application components need to be modelled to capture required information for efficient application deployments and optimisations. The information used to describe data is usually called *metadata*, i.e., data about data.

6.1 Metadata Description and Management

Two major issues are related to the description and management of metadata: (a) how they are described to summarise in the best possible way the data that they need to characterise; (b) how they are managed and stored such that they can be retrieved and exploited by the respective management system, which is DLMS in this case. In this respect, in the following, we explicate how we deal with these two issues.

In the Melodic deliverables D2.2 [6] and D2.4 [56], we explain how metadata about data are specified in the Melodic middleware. In this section, we will thus shortly summarise the way this description is modelled and structured. More details can be found in the corresponding deliverables.

The solution that has been envisaged and realised in the Melodic project involves two main aspects:

1. A schema for the metadata, *Metadata Schema,* has been produced, covering several aspects of data along with concepts that can be used to describe constraints and requirements on how a certain application can be placed on Cross-Cloud resources;
2. An extended CAMEL model that focuses on the description data and relationship between data and application components

Specifically, Metadata Schema is a taxonomy of concepts, properties and relationships that can be exploited for supporting data management as well as application deployment reasoning. In particular, this schema can be exploited, first, for specifying cloud service requirements and capabilities to support application deployment reasoning, secondly, for defining features and constraints to support data management and, third, data security-related concepts for driving the access control in the Melodic platform. The schema is clustered into three parts, which match the aforementioned three aspects, respectively.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

The first part captures the main characteristics that Cloud service offerings possess that can be used for specifying both requirements and capabilities for that services. Currently, the schema covers the two lower levels in the Cloud stack, i.e., the infrastructure and platform levels.

The second part attempts to capture those features that characterise data according to different aspects, covering data core characteristics (i.e., data volume, velocity, variety, value and quality) as well as their location, management and domains. In this sense, this part could be exploited for enriching the description of data elements, as it will be shown in the next section, that are specified in CAMEL. All such information, once specified in CAMEL, can be used to support both the management of the data specified, including the initial placement as well as its potential migration actions at runtime, when needed.

The third part attempts to cover security aspects, which can be exploited for specifying the exact context for permitting the access to data. In the Melodic context, this part of the model will be exploited as background knowledge for designing and implementing access control policies that will secure the way the Melodic platform acts on behalf of the user for placing and managing data and application components over Cross-Clouds. For more information on the Metadata Schema, the reader are referred to the Melodic deliverable D2.4 [56].

With respect to the second aspect of the Melodic modelling work, i.e., the extended CAMEL model, the overall structure of the data description has been dictated through a respective new meta-model that has been incorporated in the CAMEL multi-domain specific language (DSL). This meta-model is called *data*, as shown in Figure 3, and covers the description of data at both the type and instance level. At the type level, data and its sources are defined and can be associated with respective application components in the deployment meta-model of CAMEL dedicated to their processing and management. At this level, the structuring and composition of data can be modelled. At the CAMEL model instance level concrete data and sources can be specified which can be manipulated by respective instances of the user application, by following the pattern where one Melodic platform covers the management of the deployment and provisioning of a single user application.

In order to incorporate the actual characterisation of data and their sources, respective mechanisms have been incorporated in CAMEL to support this by drawing annotations from the metadata schema. These mechanisms include:

- The capability to extend the structure of a data model at the model level in order to incorporate the additional structure that comes with the metadata schema (in terms of further data aspects which also need to be described for a certain data element);
- The capability to incorporate additional data and data source attributes that are annotated with concepts and properties from the metadata schema. Further, existing

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

elements of CAMEL, like a whole dataset or its source, can be directly annotated with concepts from the metadata schema.

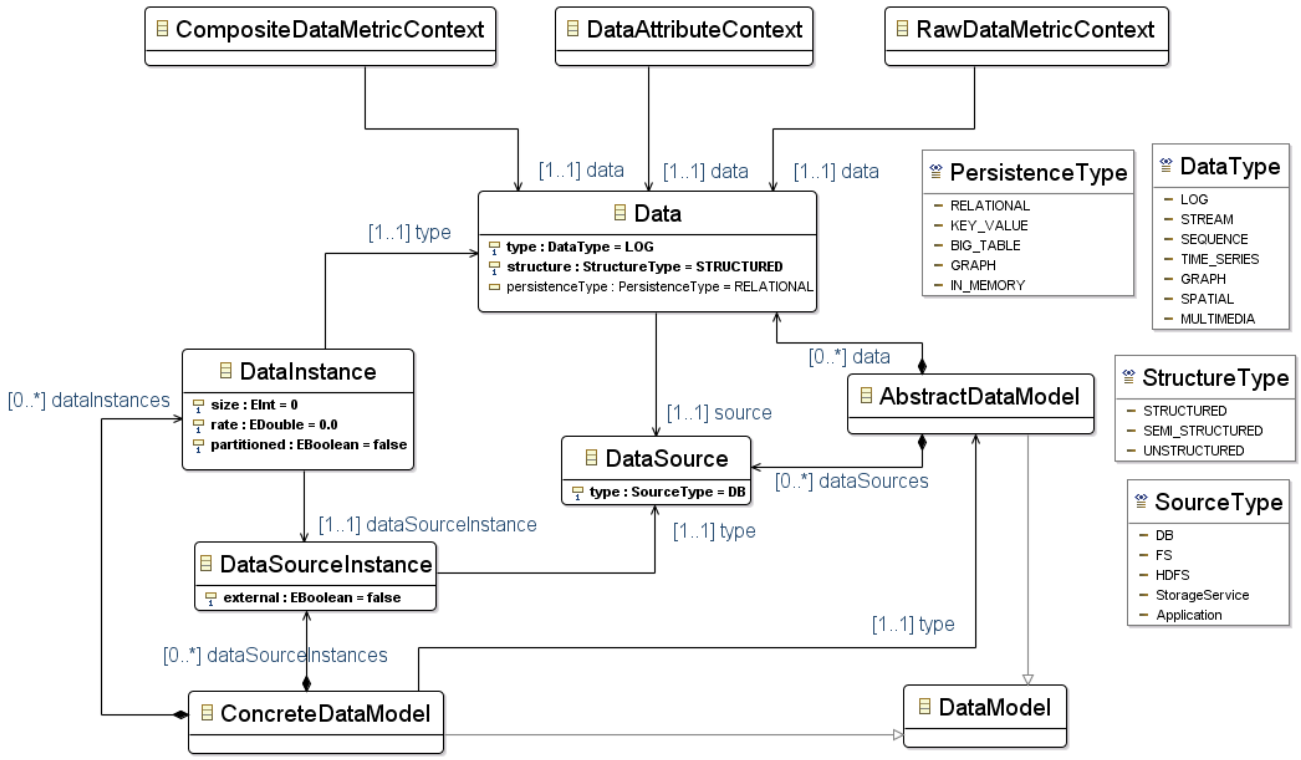


Figure 3: Data meta-model in CAMEL

Concluding this section, we note that based on the detailed classification and analysis of Data Storage and to conform to the taxonomy presented in Figure 2 (Chapter 2), we have updated the Metadata Schema presented in [56]. Specifically, the updated parts of the model with respect to the *Data Storage* class are depicted in Figure 4.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

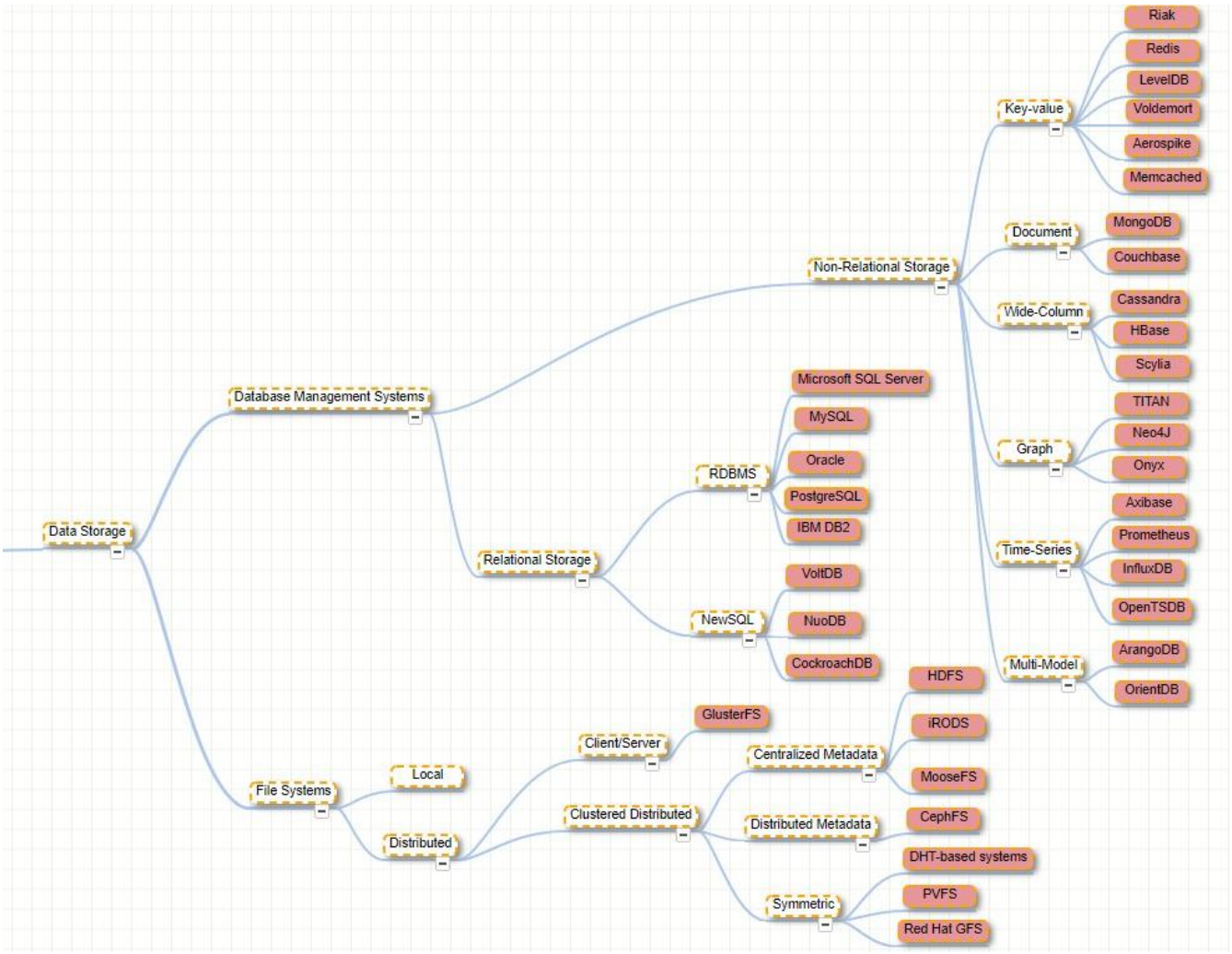


Figure 4: Updated Data Storage Class of Metadata Schema

The detailed mind map for an easier walkthrough of the updated Metadata Schema's main aspects can be found here: http://melodic.cloud/assets/images/MELODIC_Model_vFinal.png

6.2 Data Catalog

The *Data Catalog* is the place where real-time data about data instances, as well monitoring information about data access patterns of application components is stored. In other words, the *Data Catalog* is a storage medium for metadata about data sources, which enables both its storage and retrieval. In this way, Data Catalog can be regarded as the main knowledge base (KB) which needs to be consulted by the DLMS (and Upperware in general) to find out any static or dynamic information about data that can enable optimisation of the provisioned data applications across

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

different Clouds. For instance, the DLMS utilises the information stored in the Data Catalog to establish costs associated with each deployment solution.

There can be different ways and technologies via the Data Catalog can be realised. In the literature, one can see various alternatives: (a) relational database management systems (RDBMSs) which impose the structure and content of the metadata schema via the respective schema of the database that will store the content of the Data Catalog; (b) semantic KBs which conform to respective semantic standards for metadata description; (c) model repositories which store the content of the Data Catalog in the form of a model (as in the case of CAMEL).

In the context of the Melodic project, these technological alternatives have been evaluated in sight of the current solution adopted for the storage of the CAMEL models, which maps to the use of a unified model repository.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

7 Data Life-cycle Management System (DLMS)

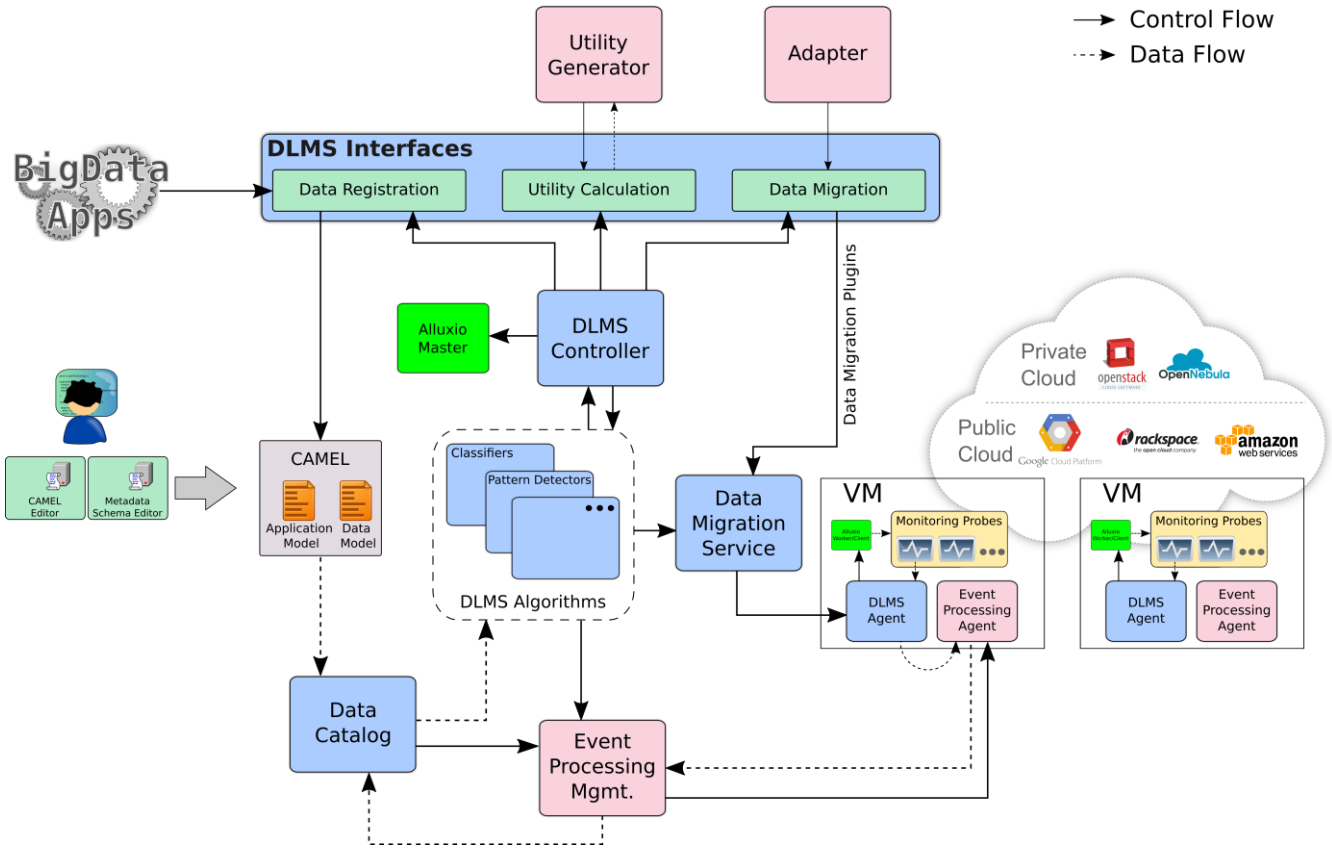


Figure 5: Overview of the DLMS Architecture

In this chapter, we describe the basic design principles and the architecture of the Data Life-cycle Management System (DLMS), and its sub-components. An overview of the DLMS architecture is presented in Figure 5.

At the heart of the DLMS is a *DLMS Controller* which manages, and coordinates with various DLMS sub-components and execute data management tasks. DLMS Controller is also responsible for periodically running various DLMS algorithms to update the internal DLMS knowledge base that is used to assign costs to the proposed deployment solutions (detailed in Chapter 8). All data sources available to the applications deployed through Melodic are modelled (see Section 6 and the Melodic deliverables D2.2 [6] and D2.4 [56]) and registered in DLMS. *DLMS Interfaces* provides interfaces for the data registration, data migration, and utility calculation using a REST based API. As soon as the user-supplied CAMEL model is available, modelling interfaces call DLMS registration API which results in reading the CAMEL model and registration of modelled data sources in the DLMS. Once a data source has been registered in DLMS, its characteristics through

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

the modelling information provided in the form of CAMEL are available for exploitation by the DLMS algorithms. Moreover, data sources registered in the DLMS are managed by the DLMS throughout their life-cycle. Note that both external and internal data sources are possible, where external data sources are referred to the ones uncontrollable by the Melodic platform and the DLMS will not be able to migrate or place them. However, all data sources, including the external ones, will be available for access by the applications through a unified namespace.

From the monitoring plane, DLMS subscribes to the data access metrics associated with the data processing by deployed application components. To obtain data access metrics, DLMS, through the Executionware, deploys *DLMS Agents* on each VM deployed by Melodic. DLMS Agents monitor data accesses between the instances of the application components and data sources and the information is eventually recorded in the Data Catalog. Making use of the historical information about the data access patterns of the application components, data source characteristics, network monitoring data between data centres, and the incurred costs in the Cloud (data access and data storage costs), DLMS assigns a utility value to all the candidate deployment solutions proposed by the solvers. The utility assignment is a multi-facet process, where the DLMS algorithms are employed, as discussed in Chapter 8. In addition to the cost assignment, when a reconfiguration to the existing deployment is required, DLMS is consulted by the Adapter component of the Upperware to trigger any data migrations required for the concerned datasets.

7.1 Approach for Data-aware Optimisations

We tackle the problem of data-aware application deployments and adaptations using a two-step approach. First, all data sources deployed by Melodic or accessed by the application components deployed by Melodic (external data sources) are modelled. The modelling enables Melodic Upperware (Solvers) to consider data-source specific location constraints and SLAs the same way as it handles the application components when calculating a candidate deployment solution. In this way, we allow solvers to propose deployment solutions irrespective of the current data locations, but eliminates any invalid solutions with respect to data constraints. The DLMS component manages data sources on behalf of the user and, thus, keeps track of current data location, historical data access patterns by different application components (through Data Catalog), and Cloud- and datacentre-dependent network performance and data transfer and access costs. The DLMS then assigns a utility value to each proposed solution, which is used by Utility Generator in the utility function for the selection of the optimal deployment solution among proposed candidate solutions.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

The CP Generator component is responsible for generating constraint programming (CP) models that are processed and solved by Melodic's solvers. The CP Generator component reads the CAMEL application model and based on it creates a CP model that expresses a constraint equation needed by solvers.

The CP Solver is responsible for solving a certain deployment reasoning/optimisation problem that is encoded in the form of a constraint model. The constraint model includes a set of constraints mappings to the SLOs defined by the user in the CAMEL model, a set of variables which denote the number of instances that individual application components and internal data components should have over a certain VM offerings that the quantitative hardware requirements posed by the user for that component. The CP Model, thus produced is solved by the CP Solver. Besides CP Solver, other optimisation solvers can also be used in the Upperware to calculate deployment solutions. A detailed Upperware architecture and component description is provided in [6].

Based on our approach, we revisit the criteria we used for the analysis of the state-of-the-art solutions in Table 2. A descriptive summary is given in Table 3.

Table 3: Evaluation Criteria and the Melodic approach

Criteria	Description
Fixed Data Positioning	CAMEL, extended by data meta-model, allows setting up constraints related to the data locations as well as any other SLOs which are handled by the solvers. Hence, the candidate solutions proposed handle them already before the assignment of the costs by the DLMS.
Constraint solving technique	The main solver used by the Upperware is based on Constraint programming model. The CP Models are extensively used to search feasible solutions from within large sets of candidate solutions by modelling the search problem in terms of arbitrary constraints [57]. The current implementation of the CP Solver relies on Choco Solver[31], which is a free constraint satisfaction optimisation programming solver.

[31] http://www.choco-solver.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Host Granularity	Fine, based on VMs
Intermediate data handling	Both fixed and intermediate data sources can be handled. However, each data source that needs to be managed by the DLMS and used in its cost assignment, should be modelled.
Multiple application handling	Multiple application components, and arbitrary relationships between the application components and data sources is handled, as described in Chapter 6.
Data size uncertainty	Handled through CAMEL. The actual of the data is not specified at the type level but at the instance level as they characterise a concrete dataset. More details are provided in Section 8.2.
Replication	As different storage technologies have encapsulated replication mechanisms, the DLMS does not target replication to avoid interfering with the internal data storage technology specific optimisations. For instance, most DFSs have a configurable property that govern the number of replicas each chunk of data will have in the cluster. Such properties can be configured at the component deployment time through user-supplied scripts.
Optimisation Criteria	A two-step based optimisation approach is used. Optimisation criteria is based on five different DLMS algorithms. Moreover, new algorithms and techniques can easily be added in the future. In addition, different *weights* can be assigned to each optimisation criteria (see Section 8.6).

7.2 Design Principles and Functionality

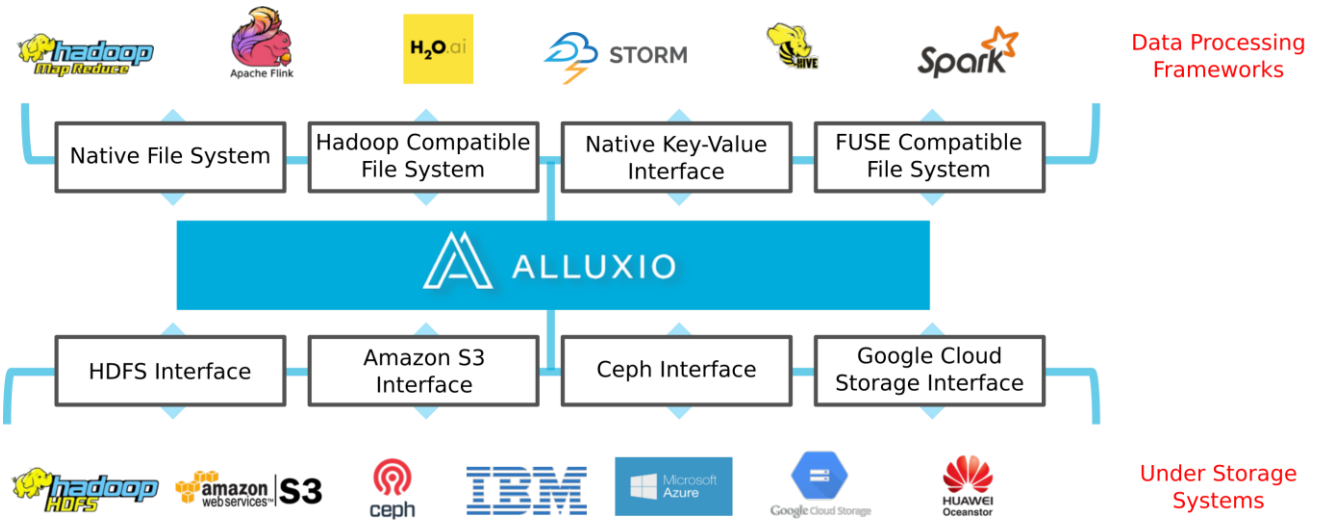
The first design principle of DLMS is that the DLMS does not take data placement and migration decisions directly, but influences the selection of a proposed solution through the assignment of a utility value to the solutions. This utility value describes how much a proposed solution is *favoured* by the DLMS, given its algorithms. Hence, all the constraints pertaining to the data sources are defined in CAMEL, and when a solution is proposed by the Solvers, these constraints are already taken care of. It means that no solution violating the user-defined data placement constraints reaches the DLMS, and hence DLMS does not validate it.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

The second important design principle of the DLMS is that all file based data sources (losscal file systems, DFSs) are available through a unified namespace accessible to all application components regardless of their locations. Finally, all DLMS algorithms relies on the information either provided by the user in CAMEL, or on the historical information obtained by monitoring the deployments. No external input is expected.

In light of these principles, DLMS offers the following functionality:

- Management of data sources on behalf of the Melodic user
- Providing a unified namespace accessible from everywhere in the Cross-Cloud deployment environment
- Assignment of a utility value to each solution proposed by the solvers
- Providing interfaces to run data-lifecycle events and data migrations
- Providing a set of standard data related metrics



7

Figure 6: Alluxio as the storage middleware, figure adopted from the Alluxio web pages

7.3 Handling DFSs

Traditionally, data-intensive applications in a cluster generally used to rely on a co-located big data compute framework and storage system. An example of such a configuration is a Hadoop MapReduce [58] running on a co-located HDFS cluster. MapReduce applications in this example will access the shared storage provided by the HDFS on distributed data processing nodes. However, as the big data ecosystem is rapidly evolving, a whole range of big data processing

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

frameworks, technologies, and storage systems have been developed in the last several years [59]. The current ecosystem involves a variety of compute frameworks, ranging from traditional Hadoop MapReduce and Apache Spark to specialised frameworks such as Apache Flink[32], Storm[33], and Samza[34], to name a few. The storage systems are equally plentiful. A variety of open source and enterprise distributed file systems, database management systems, and Cloud-specific storage technologies are in use. Moreover, modern big data applications often manage multiple data sources, requiring separate management of namespace and access APIs for each data source. In a Cross-Cloud application deployment, as targeted by Melodic, this heterogeneity of storage technologies bring interoperability issues and costly storage integrations. After evaluating the requirements of the DLMS, we decided to use Alluxio[35] (formerly Tachyon [60]) as the middleware for storage technologies. Alluxio is a rapidly growing open source memory speed virtual distributed storage system enabling big data applications to interact with data from a variety of storage systems and technologies. Alluxio has attracted a large number of active contributors, and the project is being used by a number of large companies, such as Google, Baidu, Intel, and IBM among others. Alluxio provides following salient features useful for the Melodic DLMS.

- **Unified Namespace:** Enables applications to access data across multiple data sources with different storage technologies under a single unified global namespace.
- **Support for large number of storage systems:** Alluxio supports a large number of underlying storage systems, including HDFS, Microsoft Azure Blob Store, Amazon S3, Google Cloud Storage, OpenStack Swift, GlusterFS, MaprFS, Ceph, NFS, Alibaba OSS, and Minio. Moreover, it provides a pluggable under storage system so new storage technologies can also be added on demand.
- **Performance Acceleration through Lineage:** Alluxio leverages the concept of *lineage*, to accelerate applications accessing remote data using write-caches. The lineage allows Alluxio to use write-caches without compromising on the fault tolerance. In lineage based systems, the lost output is recovered by re-executing the tasks that created the data.
- **Flexible File Access APIs:** Alluxio provides several file system interfaces, including a HDFS compatible interface to allow for easy integration of applications without the need of application code change.

[32] https://flink.apache.org/
[33] http://storm.apache.org/
[34] http://samza.apache.org/
[35] http://www.alluxio.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Figure 7 shows an example of a unified namespace offered by the DLMS through Alluxio. In the shown example, the user has access to two storage systems, an HDFS cluster with file hierarchy shown in blue, and a S3 bucket shown in red. When these data sources are registered in the DLMS, they are mounted at a location (which by convention is selected as */melodic/[DATASOURCE_NAME]* in the unified namespace. The user application can then access both data sources under a single global namespace regardless of the location of the both the data sources and application components accessing them in the Cross-Cloud.

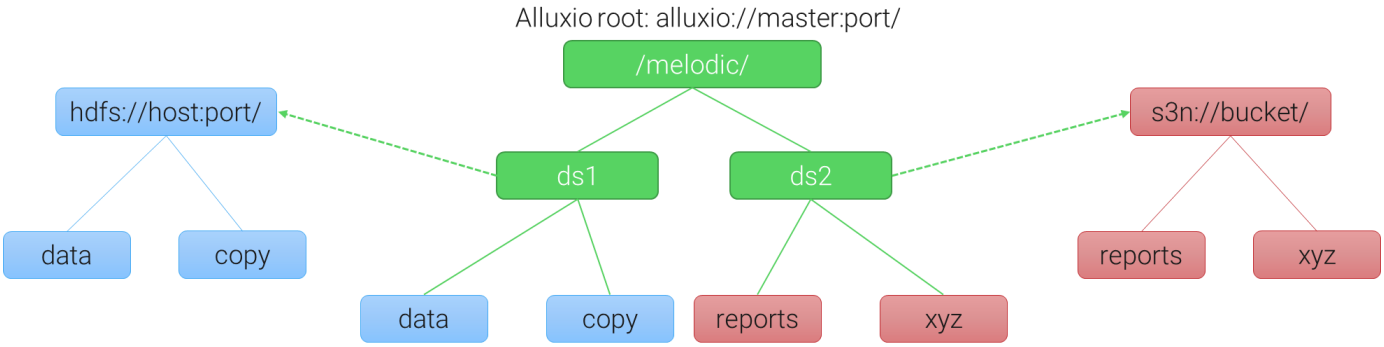


Figure 7: An example of the unified namespace offered by DLMS through Alluxio

Architecture and DLMS Integration

Alluxio is implemented using a client/server architecture, with a single primary *master* node and multiple *worker* nodes. Alluxio system is conceptually divided into three component types: the master, workers and the clients. The master and workers together create the Alluxio storage servers, while client are applications that accesses the storage systems through Alluxio. The primary master is primarily responsible for managing the global metadata of the system, whereas the secondary master replays journals written by the primary master and do periodic check-pointing to enable fault tolerance in case of a master failure. Alluxio workers are responsible for managing local resources allocated to Alluxio, such as storage disk, memory etc. As shown in Figure 8, the DLMS Controller interacts with the Alluxio master to allow for data registrations and mounting under designed places under the unified file system, while DLMS Clients access Alluxio worker nodes and Clients to obtain information required for the data access monitors for the underlying file systems. DLMS Controller and Alluxio master runs on the Melodic middleware VM, while DLMS Agents and Alluxio Workers are installed on each VM commissioned by the Executionware.

Note that, even a unified namespace access is provided to the DFSs, DFSs themselves are considered as *black boxes* for the DLMS implementation. DFS-specific configurations, such as number of replicas to be stored, caching schemes used, replication and placement policies,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

synchronization methods, and load balancing are configured directly in the respective DFSs. This is to ensure that the DLMS does not interfere with the DFS-specific features and optimisations.

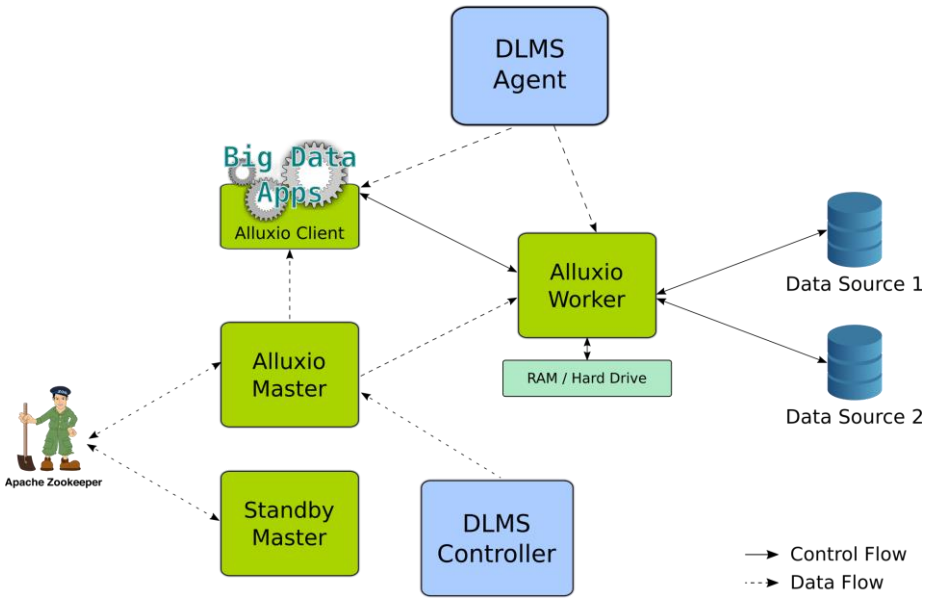


Figure 8: Interaction between DLMS and Alluxio

User Applications

As the underlying unified file system available to the applications is based on Alluxio, applications can use client libraries provided by Alluxio for file system access. As described above, we use the convention that all data sources are mounted, at the time of the data source registration, under a global namespace prefixed by `/melodic/`. So, a data source registered as name `dsource1` will be mounted at the path `/melodic/dsource1`.

The paths in alluxio follow `alluxio://server:port/` semantics. So a fully qualified URI of a file `file.txt` under `dsource1` root directory will be `alluxio://127.0.0.1:19998/melodic/dsource1/file.txt`, for example. However, when the path of alluxio master is configured via properties in the Client libraries, expansion of paths take place automatically, so relative paths such as `/melodic/dsource1/file.txt` can be used.

Alluxio provides two different Filesystem APIs, the Alluxio Filesystem API and a Hadoop compatible API. The Alluxio API provides full file system functionality, whereas, the Hadoop compatible API allows user to use application based on Hadoop without the need of code change.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

We also provide a wrapper file system interface that allows translation of the file system URIs to the unified global namespace available to the system, based on the information provided at the time of data registration in CAMEL.

7.4 Handling DBMSs

DBMSs offer encapsulated and technology-specific features for data replication, partitioning, sharding, and distribution. However, applications written for particular DBMSs can utilise Standard Query Language (SQL) interface in most DBMS implementations, as described in Chapter 3. As access to each DBMS is a specific interface, a unified global namespace is not feasible for the DBMSs. Moreover, just like the DFSs implementation, DBMSs are also considered *black-boxes* for the DLMS and specific settings are configured directly in the DBMSs, as per the user requirements.

Nevertheless, several key-value stores such as Amazon S3, OpenStack Swift[36], and databases that run on top of a DFS such as HIVE or HBASE, can be mounted on an Alluxio based interface as other DFSs.

7.5 Architecture and sub-components

DLMS offers a modular extensible architecture with each component doing a designated job under the control of the DLMS controller. As shown in the DLMS component diagram in Figure 9, Inter-Component interfaces are exposed via REST interfaces through a REST server. In particular, the *DataRegistrar* interface allows to register the data sources in the DLMS after they have been modelled, which follows *mounting* of a given data source under the unified namespace and storing the information in DLMS database sub-component. *UtilityCalculationInterface* offers a single method that is called by the Utility Generator with the proposed solution, in the form of *Node Candidates,* that is evaluated by the DLMS for cost assignments, and a utility value is returned. A *DataMigrationInterface* interface is also exposed with various methods dealing with the data migration between one data source to another. In case the files are under the same underlying file system, they can be directly migrated through the Alluxio interfaces. However, a copy and delete file system operation is needed for the data migration operations spanning multiple underlying file systems.

[36] https://docs.openstack.org/swift/latest/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

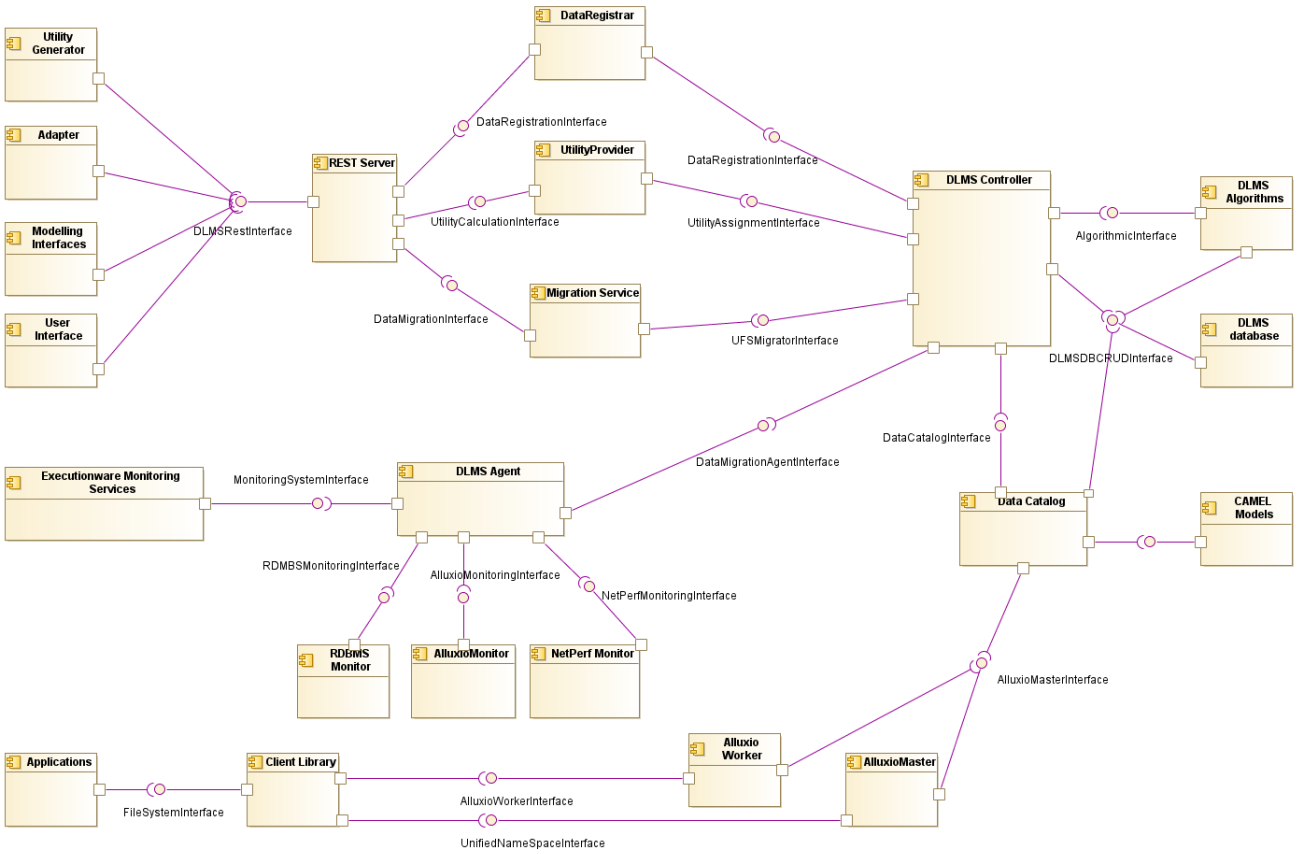


Figure 9: DLMS component diagram

The *DLMS Agents* are run on each VM resource commissioned and they provide interfaces to consume data access monitoring metrics available by the specialised data source monitors, and forward them to the Executionware monitoring services, which eventually reach Data Catalog. For all file based data sources, monitors are based on *AlluxioMonitor* that records read/write and RPC operations to the underlying file systems. In addition, a RDBMS-specific monitor is also available based on the MySQL database, in accordance with Melodic use-case requirements. However, new monitors can be added if an unsupported data storage system is used. There is also one specialised *NetPerfMonitor* which is a server/client script that runs on both ends of a communication channel between the applications and data sources (data sources that are deployed on the VMs commissioned by Melodic) to gather metrics related to the available latency and network throughput which is later utilised by the DLMS algorithms to create a *map* of available latency/throughput between different Cloud providers and data centres.

The DLMS Controller has access to the information stored in the CAMEL models, as well historical information gathered by the DLMS agents, through the Data Catalog (see Section 6.2), which is also utilised by the DLMS algorithms. Moreover, DLMS Controller also runs data migration tasks

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

on VMs through DLMS agents, when instructed by the Adapter component of the Upperware. Periodically, the DLMS Controller runs all DLMS algorithms to update their internal knowledge-base (KB) used for the cost assignments. A conceptual overview of the utility assignment by the DLMS is provided in Figure 10.

Applications interact with the data stores through a specialised client library which provides access to the alluxio-based global file system. However, client library is installed during the deployment of the application frameworks, such as Spark, so no change in the application code is needed.

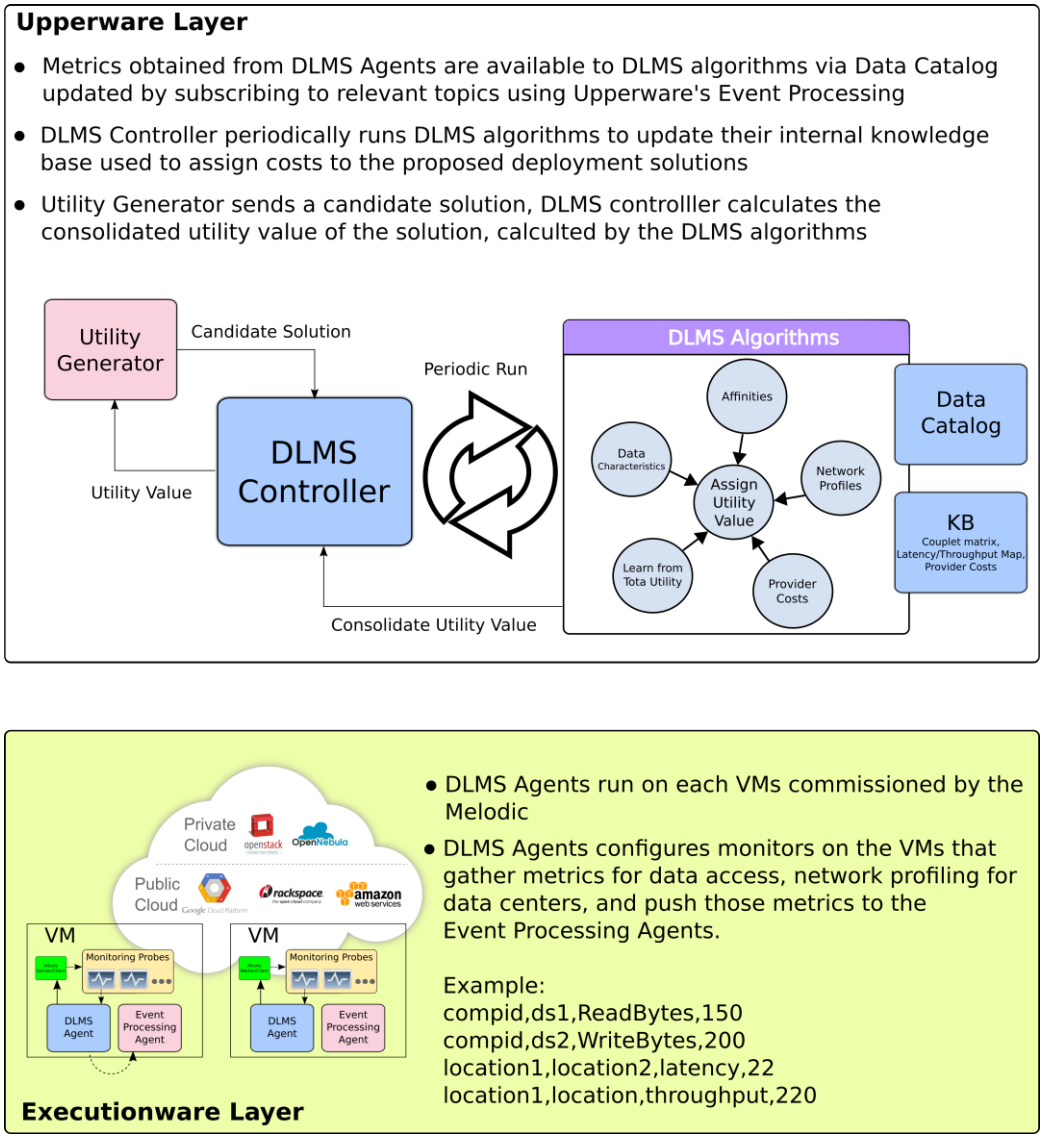


Figure 10: Conceptual overview of the utility assignment by the DLMS

7.6 DLMS Agents

DLMS Agents are installed on each node commissioned by the Melodic Executionware. The job of the DLMS Agents is to configure DLMS monitors on the nodes, and assist Data Migration Service in data migration between data sources. The Executionware configures and triggers the monitoring probes on each of the node. Based on the provided configuration, raw monitoring data from the respective nodes is provided to the *Event Processing Agents* and stored in the *MonitoringDatabase*, which is queried by the Upperware. The monitoring information gathered through the DLMS Agents is eventually stored in the Data Catalog and is utilised by the DLMS Controller and DLMS Algorithms.

Two types of monitors are configured by the DLMS Agents.

1. Monitors probing the data access from the application components to the particular data sources. These include both the metrics collected through the *Alluxio* proxy and client installed on each node as well as the *RDBMS* monitors configured according to the component specification. For instance, an important set of metrics gathers are about the amount of bytes read and written to the configured data sources from the application component(s) installed on a particular node.
2. Network performance monitors collected using *NetPerfMeter*[37]. For the internal data sources where the data source is located on VM(s) commissioned by the Melodic, the NetPerfMeter[61], [62] is a convenient and flexible open source tool developed at SRL for transport protocol performance analysis. It provides multi-platform support and works with TCP, SCTP, UDP as well as DCCP. NetPerfMeter needs a client/server installation and is able to send a saturated TCP flow between two given endpoints. During the TCP measurement, it also runs *ping* between the same endpoints to record the ICMP Echo Request/Echo Reply round trip time (RTT). In this way, both RTT and expected application payload throughput is recorded.

[37] https://github.com/dreibh/netperfmeter

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

8 DLMS Algorithms

DLMS algorithms refer to the algorithms employed by the DLMS to assign utility value to the deployment solutions. Currently, we employ five different techniques, each assigning a utility value in the interval [0 , 1] to a given deployment solution. Later, we use a weighted sum method to calculate the consolidated utility value to be returned to the Utility Generator. The DLMS algorithms employs techniques that include the use of affinities between application components and data sources, data source characteristics, network latencies and throughput between data centres, Cloud provider costs, and learning from previous DLMS decisions. Once all values are calculated, a weighted approach is used, with configurable weights given to each technique in calculating the final utility value to be delivered to the Utility Generator.

$$U_s = Normalize(a \,.\, Aff_s + b \,.\, D_s + c.N_s + d \,.\, CP_s + e \,.\, L_s)$$

Each algorithm is a *plugin* in the DLMS controller and thus, in the future, more advanced algorithms can also be developed an incorporated in the DLMS with ease.

8.1 Affinity between Application Components and Data Sources

Even though the relationship between data sources and application components is defined in CAMEL through data modelling, the actual dynamic affinities between data source instances and application component instances cannot be established in advance. In our model, the affinity between a particular application component and data source defines the degree representing how coupled they are together. The consolidated DLMS utility assignment algorithms use these affinities to assign a utility value to the proposed deployment solutions. For instance, if a solution proposes that the component *x* and data source *y* are located in geographically-distributed data centres, while the affinity between them calculated through historical information stored in the catalog is high, the DLMS algorithms are likely to assign that solution a lower value than the one which prescribes that these entities are co-located in a single data centre.

In our model, the relationship between application components and data sources, which we term as *couplet value*, is based on the profiling of data transfers (reads and writes) between them. In general, data traffic profiling is a vast topic with applications in many areas including network traffic management, anomaly detection, energy efficiency, and QoS. In our DLMS implementation, the data transfer profile of a tuple *<application_component, data_source>* is defined both by the number of data transfers, i.e. number of historical access counter with bytes transferred greater

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

than zero, and the total amount of data read and written from the data source. Number of historical points in the calculations as well as the update period is configurable through a properties file. In addition, three simple models for profiling based on linear prediction, *Equal Weights*, *Linear Weights* and *Real-time* prediction methods can be used. However, more advanced methods such as those based on autoregressive models, can easily be supported.

If t historical data reads and writes, $T^R = \{T_t^R, T_{t-1}^R \ldots, T_1^R\}$ and $T^W = \{T_t^W, T_{t-1}^W, \ldots, T_1^W\}$, are recorded between application component A and the data source D over time $\{1,2,3,\ldots t\}$ with the most recent being T_t, the total expected transfer can be calculated at time $t+1$, $\hat{T} < A, D >$ $(t+1)$ can be predicted by equation (1).

$$\hat{T}\langle A, D\rangle(t+1) = (r)\sum_{i=1}^{t} a_i T^R(t+1-i) + (1-r)\sum_{i=1}^{t} a_i T^W(t+1-i) \quad (1)$$

Where r is the preference value given to the data read which could be 0.5 in case of no preference over writes (or same costs for reads and writes). The predictor coefficients, a_i, represent weights historical points received in predicting total transfer according to their place in the time series. In equal weights profiling, each historical counter (which is an entry of historical record) has the same weight in predicting future traffic, whereas, the linear network profiling uses a linear function to decrease the weight of each counter further we go in the time series. The third method we currently support is based on real-time data counters. In real-time network prediction, the traffic profiles are calculated based on the most recent data counter recorded.

$$a_i = \begin{cases} n & \text{if equal weights for data counters} \\ i & \text{if linear weights for data counters} \\ 1 & \text{if real-time prediction is used and i = t} \\ 0 & \text{if real-time prediction is used and i != t} \end{cases} \quad (2)$$

Once all the values for non-normalized expected transfer between each *<application_component, data source>* have been calculated, normalized values between range $[a, b]$ can be assigned using linear transformation. In our case, we choose the range $[0, 1]$. Moreover, the maximum and minimum values can be self-assigned as well to reduce chances of skewed normalized values in case no application component has strong relationship with any data source in the application.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

$$T\langle A, D\rangle = (x - a)\frac{b - a}{\max(\hat{T}) - \min(\hat{T})} + a, \forall x \in \hat{T} \quad (3)$$

The same process can be followed to get normalized values for the number of data transfers in the range [0, 1] from the historical data transfer counters for both read and writes.

$$\hat{K}\langle A, D\rangle(t+1) = (r)\sum_{i=1}^{t} a_i K^R(t+1-i) + (1-r)\sum_{i=1}^{t} a_i K^W(t+1-i) \quad (4)$$

$$K\langle A, D\rangle = (x - a)\frac{b - a}{\max(\hat{K}) - \min(\hat{K})} + a, \forall x \in \hat{K} \quad (5)$$

Finally, a weighted sum can be applied to calculate the final couple value for each <application component, data_source> tuple.

$$Couplet\langle A, D\rangle = (w)T\langle A, D\rangle + (1 - w)K\langle A, D\rangle \quad (6)$$

The result in a couplet value table in the form of a matrix as shown in Figure 11.

Couplet Value Table

com	ds1	ds2		ds_n
1001	0.22	0.52		0.42
1002	0.00	0.21		0.11
1003	0.01	0.12	....	0.00
....				
com_{n-1}	1.00	0.61		0.00
com_n	0.92	0.98		0.00

Figure 11: An example of a couplet value table

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

The total affinity value of a solution, provided as a solution with actual number of instances of application components and data sources can be calculated by summing the individual affinities and normalising to get a value in the range [0 , 1]. Pseudo-code for this simple sum-based solution is given in the listing below.

Listing ASSIGN TOTAL AFFINITY VALUE

Require: Couplet value matrix, M
Require: CAMEL models for the application
Require: max, min for output in range $[0, 1]$
1: $affinity \leftarrow 0$
2: **for each** $com \in appComponents[]$ **do**
3: **for each** $ds \in dataSources[]$ **do**
4: **if** getType(com) relates to getType(ds) in CAMEL **then**
5: $affinity \leftarrow affinity + M(com, ds)$
6: **end if**
7: **end for**
8: **end for**
9: $normalized \leftarrow affinity.\frac{1}{max-min}$

8.2 Data Source Characteristics

Data source characteristics are gathered through CAMEL, as well as from size and capacity probe from the DLMS monitors. Based on the data source size, a utility value is assigned to each migration requested in the proposed solution by simple linear transformation. However, size is not the only data source characteristic that can be taken into consideration for assigning utility. For instance, data source type can also reflect how costly the migration would be, given complex configurations required for some data sources. This is marked as a future exploration area.

To explain the use of data source characteristics, we provide a simple example. Suppose that we have an application that processes images. These images map to a specific dataset, which is continuously increased with a rate of 100 images per day (500 MB per day, as one image is about 5 MB in size). The original size of the dataset is 1 GB.

The respective dataset in CAMEL will be modelled via a certain *Data* element, as show in Figure 3, which will be annotated directly with respective concepts from the metadata schema. In particular, this element will be annotated with the *File* concept from the metadata schema, which is sub-concept of *Format*. In this way, we annotate the data set with feature that characterise its format.

Now, the actual size and increase rate of the data set will not be specified at the type but at the instance level as they characterise a concrete dataset, which will be actually processed by the user application. In this respect, we will specify a certain *DataInstance* element having a type

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

which maps directly to the *Data* element that has been created. For this data instance, CAMEL does not cover directly additional information but allows to enrich its description with extra information from the metadata schema. As such, one *Attribute* and one *Feature* (i.e., the element in CAMEL that enables the extension of the structure) will be inserted to the data instance element: (a) an attribute with the name 'size', which is annotated with the *hasSize* property from the metadata schema and has the value of 1. This attribute is also associated with the unit of GBs; (b) a feature named as size, which is annotated with the *Size* concept from the metadata schema.

8.3 Network Monitoring

RTT times and application payload throughput gathered through NetPerfMeter are used to create a latency and throughput map between data centres. When no information is available, the latency and available throughput map can be initialized based on whether the two data centre locations are from the same cloud provider as well as based on their geographical coordinates. However, with the each new deployment on a different Cloud provider / data centre, network monitoring metrics gathered through the NetPerfMonitors will result in new entries to the network monitoring map created by the DLMS. The network monitoring map, in the form of a matrix, is then used to assign a network performance utility value based on the proposed solution, prescribing locations of the application components and data sources.

8.4 Cloud Providers Costs

Data transfer from one data centre to another results in a monetary cost associated with the Cloud provider fees. The actual incurred costs depends on Cloud providers in question, amount of data being transferred, and the source and destination data centres. A comprehensive cost modelling of Cloud providers have proven to be difficult, and depends on a large number factors. Some Cloud providers, for instance Microsoft Azure, do not charge data transfers between their own data centres, while others may charge different prices for data transfers based on geographical-locations of the data centres. In addition, data transfer between private Clouds may not cost direct transfer fees. Moreover, not all information pertaining to the prices can be dynamically loaded from the Cloud provider APIs, rather the reverse is true. In such situation, a generalised approach can be used to assign utility values to the data transfers. For instance, a utility of 1 can be issued if data is transferred in the same data centre of the same Cloud provider, while a utility of 0 can be designated when data is being transferred across the data centres at different geographical locations owned by different Cloud providers. In the initial DLMS implementation, we use a simple

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

algorithm to assign utility value to the data transfers, based on a parameter called *distance* between different locations, as shown below in the pseudo-code.

Listing ASSIGN A COST TO DATA TRANSFER

```
Require: A src and dst data centre
Require: Amount of data being transferred, size
dist ← 1
if src.CloudProvider and dst.CloudProvider are not private clouds then
    if src.CloudProvider != dst.CloudProvider then
        dist ← dist + 10
    else
        check the region
        if src.Region != dst.Region then
            dist ← dist + 5
        end if
    end if
end if
// both for private clouds and public clouds
if src.GeographicalRegion != dst.GeographicalRegion then
    dist ← dist + 2
end if
transfer ← size * distance
// linear transformation to set cost using distance in
// interval [0 , 1] based on [min,max]
```

8.5 Learning From Previous Decisions

DLMS utilises the information available through the Data Catalog and uses algorithms to determine the cost values for each proposed solution. However, the actual application performance resulted from a decision taken by the DLMS, as perceived by the application, can only be reflected by the overall utility calculated by the Utility Generator [6]. In order to take that into consideration, and learn from the decisions made by the DLMS, a specialised sensor is installed with the value reporting the overall utility of the currently deployed solution. DLMS subscribes to this metric and use it find out impact of the previous DLMS decisions on the overall application utility.

To formulate this problem, each selected deployment solution is mapped as a Graph $G(V,E)$, where vertices V represents both the component instances and datasets. However, as location (which is drawn as **CloudProvider.Region.GeographicalLocation**) is associated with each component instance, the graph expands component instances into two nodes indicating a coupling of component instance and its location. Same is done for the data source instances. Example graph thus created is shown in Figure 12. Each selected solution is thus mapped into a graph and annotated with the total utility value obtained from the sensor. Whenever, a new candidate solution is proposed, we use graph similarity to find which of the previous proposed solution it is closest to, and based on the similarity and the total utility value associated with that solution, we assign a value in the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

range [0 , 1], where 1 indicate a solution that yielded the best application utility among the deployed solutions.

Graph similarity has been well studied topic in the literature and different techniques have been proposed. Most graph similarity solutions can be classified into three types, the algorithms based on graph isomorphism [63][64], feature extraction [65], and iterative methods [66] [67] [68] [69] [70]. The key idea behind these methods is that similar graphs share certain properties, such as degree distribution, node sub-graphs, that can be used to assign a similarity value to two graphs.

In our implementation based on [67], we use the original Belief propagation algorithm, which is an iterative algorithm, to find the similarity between the graphs created from the deployment solutions.

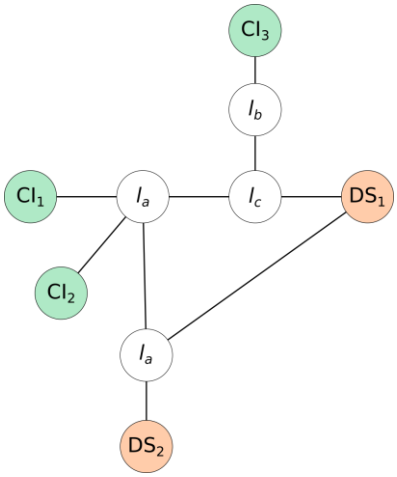


Figure 12: Graph from an example deployment solution

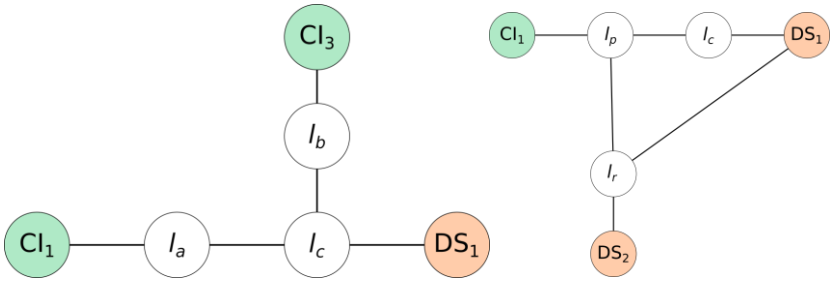


Figure 13: Example candidate deployment solutions for graph similarity

9 Conclusions and Future Work

In this deliverable, we presented a state-of-the-art analysis of existing data placement and migration methodologies, relevant for the Cross-Cloud data-intensive computing. Based on our analysis, we deduced that the existing data placement techniques are not sufficient for the Cross-Cloud application deployments in the context of Melodic. We then proceed to report on the research and design of a dedicated Upperware component, DLMS, targeted to complement Melodic with the data-aware application deployments. The DLMS component, manages data sources on behalf of the user and, thus, keeps track of current data location, historical data access patterns by different application components, and Cloud- and datacentre-dependent network performance and data transfer and access costs. The DLMS then assigns a utility value to each proposed solution, which is used by Utility Generator in the utility function for the selection of the optimal deployment solution among proposed candidate solutions.

Future work includes designing more advanced cost assignment algorithms for the DLMS utilising recent advances in machine learning. Further, the DLMS component needs to be integrated with the Upperware workflow in the Release 2.0 of the Melodic platform. In addition, a comprehensive analysis of the proposed solutions in the real-world settings is necessary to warrant desired functionality and adjustments in the proposed algorithms.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

References

[1] T. McGuire, J. Manyika, and M. Chui, "Why big data is the new competitive advantage," *Ivey Bus. J.*, vol. 76, no. 4, pp. 1–4, 2012.
[2] "What is Data Gravity? - Definition from Techopedia." [Online]. Available: https://www.techopedia.com/definition/28768/data-gravity. [Accessed: 14-Jun-2018].
[3] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards Trusted Cloud Computing," in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, Berkeley, CA, USA, 2009.
[4] R. K. L. Ko *et al.*, "TrustCloud: A Framework for Accountability and Trust in Cloud Computing," in *2011 IEEE World Congress on Services*, 2011, pp. 584–588.
[5] J. O. Kephart and R. Das, "Achieving self-management via utility functions," *IEEE Internet Comput.*, vol. 11, no. 1, 2007.
[6] Y. Verginadis *et al.*, "D2.2 Architecture and Initial Feature Definitions." 2018.
[7] Y. Verginadis *et al.*, "D2.1 System Specification." The Melodic H2020 Project Deliverable D2.1, 2017.
[8] F. Zahid, "D3.2 Business logic for supporting the complete data and data-intensive application life-cycle management." 2018.
[9] V. Turner, J. F. Gantz, D. Reinsel, and S. Minton, "The digital universe of opportunities: Rich data and the increasing value of the internet of things," *IDC Anal. Future*, 2014.
[10] D. Laney, "3D data management: Controlling data volume, velocity and variety," *META Group Res. Note*, vol. 6, p. 70, 2001.
[11] T. Lukoianova and V. L. Rubin, "Veracity roadmap: Is big data objective, truthful and credible?," 2014.
[12] I. D. Constantiou and J. Kallinikos, "New games, new rules: big data and the changing context of strategy," *J. Inf. Technol.*, vol. 30, no. 1, pp. 44–57, 2015.
[13] F. Zahid, "Network Optimization for High Performance Cloud Computing," PhD Thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2017.
[14] N. Herbst, S. Kounev, and R. Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not," 2013. .
[15] D. Agrawal, A. E. Abbadi, S. Das, and A. J. Elmore, "Database Scalability, Elasticity, and Autonomy in the Cloud," in *Database Systems for Advanced Applications*, Springer, Berlin, Heidelberg, 2011, pp. 2–15.
[16] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.
[17] B. Depardon, G. L. Mahec, and C. Séguin, "Analysis of Six Distributed File Systems," report, Feb. 2013.
[18] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Commun. Surv. Tutor.*, vol. 13, no. 3, pp. 311–336, 2011.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

[19] D. J. Abadi, "Data management in the cloud: Limitations and opportunities," *IEEE Data Eng Bull*, vol. 32, no. 1, pp. 3–12, 2009.
[20] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.
[21] D. Pritchett, "BASE: An ACID alternative," *Queue*, vol. 6, no. 3, pp. 48–55, 2008.
[22] K. Grolinger, W. A. Higashino, A. Tiwari, and M. A. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *J. Cloud Comput. Adv. Syst. Appl.*, vol. 2, no. 1, p. 22, 2013.
[23] L. Wu, L. Yuan, and J. You, "Survey of large-scale data management systems for big data applications," *J. Comput. Sci. Technol.*, vol. 30, no. 1, pp. 163–183, 2015.
[24] S. Sakr, "Cloud-hosted databases: technologies, challenges and opportunities," *Clust. Comput.*, vol. 17, no. 2, pp. 487–502, 2014.
[25] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *Acm Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.
[26] D. Pritchett, "Base: An acid alternative," *Queue*, vol. 6, no. 3, pp. 48–55, 2008.
[27] F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter, "NoSQL database systems: a survey and decision guidance," *Comput. Sci.-Res. Dev.*, vol. 32, no. 3–4, pp. 353–365, 2017.
[28] A. Pavlo and M. Aslett, "What's really new with NewSQL?," *ACM Sigmod Rec.*, vol. 45, no. 2, pp. 45–55, 2016.
[29] J. C. Corbett *et al.*, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst. TOCS*, vol. 31, no. 3, p. 8, 2013.
[30] "The design philosophy of the G-EXEC system - ScienceDirect." [Online]. Available: https://www.sciencedirect.com/science/article/pii/0098300476900649. [Accessed: 14-Jun-2018].
[31] G. DeCandia *et al.*, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS operating systems review*, 2007, vol. 41, pp. 205–220.
[32] F. Chang *et al.*, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst. TOCS*, vol. 26, no. 2, p. 4, 2008.
[33] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.
[34] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Time Series Management Systems: A Survey," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 11, pp. 2581–2600, 2017.
[35] E. Levy and A. Silberschatz, "Distributed File Systems: Concepts and Examples," *ACM Comput Surv*, vol. 22, no. 4, pp. 321–374, Dec. 1990.
[36] S. Androutsellis-theotokis, *A Survey of Peer-to-Peer File Sharing Technologies*. 2002.
[37] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, "A survey of peer-to-peer storage techniques for distributed file systems," in *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, 2005, vol. 2, pp. 205-213 Vol. 2.
[38] R. B. Ross, R. Thakur, and others, "PVFS: A parallel file system for Linux clusters," in *Proceedings of the 4th annual Linux showcase and conference*, 2000, pp. 391–430.
[39] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen, "Ivy: A read/write peer-to-peer file system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 31–44, 2002.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

[40] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Gener. Comp Syst*, vol. 26, no. 8, pp. 1200–1214, 2010.
[41] M. Ebrahimi, A. Mohan, A. Kashlev, and S. Lu, "BDAP: A Big Data Placement Strategy for Cloud-Based Scientific Workflows," in *BigDataService*, 2015, pp. 105–114.
[42] Q. Xu, Z. Xu, and T. Wang, "A Data-Placement Strategy Based on Genetic Algorithm in Cloud Computing," *Int. J. Intell. Sci.*, vol. 5, no. 3, 2015.
[43] M. LeBeane, S. Song, R. Panda, J. H. Ryoo, and L. K. John, "Data Partitioning Strategies for Graph Workloads on Heterogeneous Clusters," in *SC*, Austin, Texas, 2015, pp. 1–12.
[44] U. V. Catalyurek, K. Kaya, and B. Uçar, "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, 2011, pp. 45–54.
[45] Ü. Çatalyürek and C. Aykanat, "PaToH (partitioning tool for hypergraphs)," in *Encyclopedia of Parallel Computing*, Springer, 2011, pp. 1479–1487.
[46] B. Yu and J. Pan, "Location-aware associated data placement for geo-distributed data-intensive applications," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 603–611.
[47] J. Zhang, J. Chen, J. Luo, and A. Song, "Efficient Location-Aware Data Placement for Data-Intensive Applications in Geo-distributed Scientific Data Centers," *Tsinghua Sci. Technol.*, vol. 21, no. 5, pp. 471–481, 2016.
[48] K. Lan, S. Fong, W. Song, A. V. Vasilakos, and R. C. Millham, "Self-Adaptive Pre-Processing Methodology for Big Data Stream Mining in Internet of Things Environmental Sensor Monitoring," *Symmetry*, vol. 9, no. 10, p. 244, Oct. 2017.
[49] K. A. Kumar, A. Deshpande, and S. Khuller, "Data Placement and Replica Selection for Improving Co-location in Distributed Environments," *CoRR*, vol. abs/1302.4168, 2013.
[50] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated Data Placement for Geo-Distributed Cloud Services," *Microsoft Res.*, Apr. 2010.
[51] T. G. Papaioannou, N. Bonvin, and K. Aberer, "Scalia: An adaptive scheme for efficient multi-cloud storage," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, 2012, pp. 1–10.
[52] C.-H. Hsu, K. D. Slagter, and Y.-C. Chung, "Locality and loading aware virtual machine mapping techniques for optimizing communications in MapReduce applications," *Future Gener. Comput. Syst.*, vol. 53, pp. 43–54, Dec. 2015.
[53] J. Wang, Q. Xiao, J. Yin, and P. Shang, "DRAW: A New Data-gRouping-AWare Data Placement Scheme for Data Intensive Applications With Interest Locality," *IEEE Trans. Magn.*, vol. 49, no. 6, pp. 2514–2520, Jun. 2013.
[54] A. Quamar, K. A. Kumar, and A. Deshpande, "SWORD: Scalable Workload-aware Data Placement for Transactional Workloads," in *Proceedings of the 16th International Conference on Extending Database Technology*, New York, NY, USA, 2013, pp. 430–441.
[55] A. Rafique, D. V. Landuyt, B. Lagaisse, and W. Joosen, "Policy-Driven Data Management Middleware for Multi-cloud Storage in Multi-tenant SaaS," in *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, 2015, pp. 78–84.
[56] Y. Verginadis, I. Patiniotakis, C. Halaris, G. Mentzas, K. Kritikos, and K. Jeffery, "D2.4 Metadata Schema." The Melodic H2020 Project Deliverable D2.4, 2017.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

[57] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*. Elsevier, 2006.
[58] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[59] C. L. Philip Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," *Inf. Sci.*, vol. 275, no. Supplement C, pp. 314–347, Aug. 2014.
[60] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, Memory Speed Storage for Cluster Computing Frameworks," in *Proceedings of the ACM Symposium on Cloud Computing*, New York, NY, USA, 2014, pp. 6:1–6:15.
[61] T. Dreibholz, "NetPerfMeter: A Network Performance Metering Tool," *Multipath TCP Blog*, Sep. 2015.
[62] T. Dreibholz, M. Becke, H. Adhari, and E. P. Rathgeb, "Evaluation of A New Multipath Congestion Control Scheme using the NetPerfMeter Tool-Chain," in *Proceedings of the 19th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Hvar, Dalmacija/Croatia, 2011, pp. 1–6.
[63] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism," *J ACM*, vol. 23, no. 1, pp. 31–42, Jan. 1976.
[64] "A graph distance metric combining maximum common subgraph and minimum common supergraph - ScienceDirect." [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865501000174. [Accessed: 18-Jun-2018].
[65] M. Pelillo, "Matching free trees, maximal cliques, and monotone game dynamics," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 11, pp. 1535–1541, Nov. 2002.
[66] D. Koutra, A. Parikh, A. Ramdas, and J. Xiang, "Algorithms for Graph Similarity and Subgraph Matching," p. 50.
[67] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," *Explor. Artif. Intell. New Millenn.*, vol. 8, pp. 236–239, 2003.
[68] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity flooding: a versatile graph matching algorithm and its application to schema matching," in *Proceedings 18th International Conference on Data Engineering*, 2002, pp. 117–128.
[69] G. Jeh and J. Widom, "SimRank: A Measure of Structural-context Similarity," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2002, pp. 538–543.
[70] L. A. Zager and G. C. Verghese, "Graph similarity scoring and matching," *Appl. Math. Lett.*, vol. 21, no. 1, pp. 86–94, Jan. 2008.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664