Baltic J. Modern Computing, Vol. 8 (2020), No. 1, pp. 21–47
https://doi.org/10.22364/bjmc.2020.8.1.02

Handling Evolution in Big Data Architectures

Darja SOLODOVNIKOVA, Laila NIEDRITE

Faculty of Computing, University of Latvia, Riga, Latvia

{darja.solodovnikova, laila.niedrite}@lu.lv

Abstract. One of the purposes of Big Data systems is to support analysis of data gathered from heterogeneous data sources. Since data warehouses have been used to achieve the same goal, they could be leveraged also to provide analysis of Big Data. The problem of adapting data warehouse data and schemata to changes in user requirements and data sources has been studied by many researchers worldwide. However, innovative methods must be developed also to support evolution in Big Data systems. In this paper, we analyze architectures designed for Big Data processing and analysis described in the literature with the purpose to identify the most appropriate solution for the evolution problem. We concentrate on four architecture types: data lakes, virtual integration, polystores, and λ-architecture, and, in addition to them, we consider solutions that apply data warehouse/OLAP methods to Big Data processing and analysis. Finally, we describe our proposal of an architecture that allows to perform different kinds of analytical tasks on Big Data retrieved from multiple heterogeneous data sources with different latency and is capable of processing changes in data sources as well as evolving analysis requirements.

Keywords: Big Data, Architecture, Data Warehouse, Metadata, Evolution

1 Introduction

Data warehouses (DW) and OLAP methods have been used for several decades to support analysis of structured data sets and, therefore, many solutions to known research problems have been developed in the context of traditional relational database environments. One of such problems is a data warehouse evolution that occurs due to changes in business requirements or data sources or improvements of a data warehouse design. There are in general two approaches to solving evolution problems. One approach is to adapt just the existing data warehouse schema (Bentayeb et al., 2008) or ETL processes (Wojciechowski, 2018) without keeping the history of changes and another approach (Ahmed et al., 2014; Golfarelli et al., 2006; Malinowski and Zimnyi, 2008) is to maintain multiple versions of schema that are valid during some period of time. Futhermore, several studies (Thakur and Gosain, 2011; Thenmozhi and Vivekanandan, 2014; Solodovnikova et al., 2015) propose solutions to the formalization of requirements of a data warehouse and treatment of their evolution.

Recently, due to the increase in the volume and heterogeneity of data that needs to be processed and analyzed, Big Data technologies (Shvachko et al., 2010) have emerged that use distributed data storage methods and process data in parallel. The demand to analyze data stored in such systems is increasing and one of the analysis options is to use data warehousing. Open Big Data challenges and research directions have been outlined in several recent articles (Abaker et al., 2014; Ceravolo et al., 2018; Cuzzocrea et al., 2013; Kaisler et al., 2013; Holubov et al., 2019). The authors of the paper (Kaisler et al., 2013) mention dynamic design challenges for big data applications, which include data expansion that occurs when data becomes more detailed. Another review paper (Cuzzocrea et al., 2013) is more specific and indicates research directions in the field of data warehousing and OLAP. Among others, the authors also mention the problem of designing OLAP cubes according to user requirements. In the paper (Ceravolo et al., 2018) the authors state that detecting data and structural changes and handling them in a Big Data system is a complex task that requires further research. Another recent vision paper (Holubov et al., 2019) analyses research challenges in multi-model environments and suggests that efficient management of schema evolution and propogation of schema changes to affected parts of the system is a complex task and one of the topical issues.

One of the decisions that must be made when solutions to the evolution problems are developed is regarding the choice of the appropriate software architecture for Big Data processing and analysis. For now the research articles and practical projects offer architectures that mainly can be classified into four groups (Stefanowski et al., 2017): data lakes, virtual integration architectures, polystores, and λ-architecture (Marz and Warren, 2013).

Data Lake is for now the most popular architecture type. The data lake supports a physical integration of big data. All data produced in an organization are collected in one single place without any processing. Data Lake stores heterogeneous data in their original formats and granularity. Because of data loads from the sources to the lake, keeping the data up to date becomes a problem in the Data Lake. Another problem is to provide to the user the analysis possibility over heterogeneous data, what causes well-known problems from the data integration field, for example data semantics, data quality and others. These problems become more complex in the Data Lake because of characteristics of big data. Metadata can help to find solutions for these problems.

Virtual architecture supports virtual integration; it is not intended for physical data transfer from sources to a common repository. The architecture provides a global view over the data sources. Because the data are left in the data stores, they have not latency problems, however, different problems become topical, for example, the queries in such virtual architecture are more slowly and complex.

Polystore architecture is built specially for Big Data, unlike the data lake that has similarities with data warehouses or virtual architecture that can be considered as an inheritor of federated architectures from 90-ties. Polystore was introduced in 2015 (Duggan et al., 2015). The main idea is that the architecture consists of islands that are constructs for storing data of the same type, for example, relational island stores relational data, but text data are stored in text island. The island has a common data model and query language. The data, when it is optimal and technically supported, can be moved

from one island to another. More detailed information about this architecture type is given in this paper in description of particular systems from this architecture type.

λ-architecture is composed of three layers. In the batch layer massive volumes of data are periodically obtained from data sources and batch views are precomputed on source data. Processed data from these batch views is accessible to queries via the serving layer. Since batch views are augmented only periodically and real-time data may be necessary for analysis, the speed layer is responsible for ingestion and processing of data that is missing in batch views.

The implementation of different architecture types for big data usually provide the data storage in distributed file systems, for example HDFS, and the data parallel processing by Map Reduce. However, these architecture types are quite different, therefore particular components of architectures can be implemented by means of very different techniques, tools, methods, databases etc. Technology vendors such as IBM, Oracle, SAP, and Microsoft provide cloud and on-premise platforms for management and analysis of Big Data, but in our study, we concentrate on selective solutions based on open source tools and technologies.

The contribution of this paper is twofold. First, it performs a comparative analysis of selective existing solutions to Big Data processing and analysis available in the literature with the purpose to evaluate the appropriateness of existing Big Data approaches for the development of a system with evolution support. For each approach, the paper discusses a chosen architecture, support of Big Data evolution, metadata and technologies used in the implementation. Second, we propose an architecture that allows to store and process structured and unstructured data at different levels of detail, analyze them using OLAP capabilities and semi-automatically manage changes in requirements and data expansion. The operation of the architecture components responsible for OLAP analysis and evolution handling is based on the metadata described in the paper.

The rest of the paper is organized as follows. In Section 2 our research questions are discussed. In Section 3 the architectures analyzed in this study are classified and explained in detail. In Section 4 the approaches applied in the architectures to support evolution and metadata that are used in each architecture are discussed. We conclude with directions for future work in Section 4.

2 Research Questions

This paper aims to explore architectures that are used for Big Data processing with the purpose to evaluate their abilities to provide evolution support. A number of research questions are raised in our study to understand the composition of different architectures, including the use of data warehouses and metadata, and whether solutions to support evolution are included into these architectures.

RQ1 What specific architectures are proposed for each type of Big Data architectures, which solutions do they leverage? For the research, we selected papers that propose distinctive architectures corresponding to architecture type classification defined in the paper (Stefanowski et al., 2017): data lakes, virtual integration architectures, and polystores.

RQ2 What is the role of data warehouses in Big Data architectures? One of the options for architecture development for evolution support is to use the experience and solutions that exist in a data warehousing field, so it is also necessary to determine whether data warehouses are incorporated into Big Data architectures. Therefore, papers that propose data warehouse as an architecture component were also selected for our study.

RQ3 How evolution is understood in Big Data architectures, which methods are proposed to support evolution? In our study, we also explored architectures that were discussed in the review section of the paper (Nadal et al., 2017), where architectures were analyzed based on different Big Data characteristics, including Variety, which is one of the aspects that determines the need for evolution support. Three architectures were added to the list of the studied architectures, which support certain evolution aspects, according to the authors of the paper (Nadal et al., 2017).

RQ4 What is the role of metadata in Big Data architectures? RQ4 was studied for all architectures selected for research, additional papers were not included in the list of the examined architectures.

3 Architectures for Big Data Analysis

To answer our first research question RQ1, we classified all examined studies into four architecture types: BDW-Big Data warehouse/OLAP type, DL-data lake, VI-virtual integration architecture, P-Polystore, as well as O-other variations of standart types. Table 1 outlines the references explored in our study and clusters solutions into the mentined architecture types.

In relation to the second review question RQ2, the role of a data warehouse is also indicated if such component is employed in the architecture. Analysing the data in the table 1, an observation can be made that a data warehouse is either a central component of the proposed architecture or it is feeded with data from a data lake. Solutions that follow principles of the virtual integration, polystore or λ-architecture do not include a data warehouse component.

Table 1: Architectures for Big Data analysis

Paper	**Architecture**	**DW Role in the Architecture**	**Batch/Stream Processing**
(Chen, 2010)	BDW	DW is the central component of the architecture.	Batch
(Chen et al., 2017)	BDW (pre-processed data is stored in the data storage module)	The system transforms raw data into pre-calculated data cubes and encompasses OLAP analysis module.	Batch& stream

Paper	Architecture	DW Role in the Architecture	Batch/Stream Processing
(Santos et al., 2017)	BDW	DW is the central component of the architecture. The data in a DW is loaded from the Big Data staging area.	Batch
(Song et al., 2015)	BDW	DW is the main system component, which stores data in HDFS.	Processing mode depends on user-defined ETL process
(Sumbaly et al., 2013; Wu et al., 2012)	BDW	Data from the source is transformed into OLAP cubes.	Batch (every 2 hours)
(Tardio et al., 2015)	BDW	DW is the central component of the architecture	Batch & stream
(Dobson et al., 2018)	DL	Data from a data lake is periodically loaded into a DW by ETL processes	Batch & stream
(Zhuang et al., 2016)	DL (unstructured data management system)	N/A	Batch and real-time task processing
(Hai et al., 2016)	DL	N/A	It can be inferred that both batch and stream modes are possible
(ElSheikh et al., 2013)	VI	N/A	Batch & stream
(Yuan et al., 2010)	VI	N/A	Batch & stream
(Gadepally et al., 2017)	P	N/A	Batch
(Wang et al., 2017)	P	N/A	Batch
(Alsubaiee et al., 2014)	O (local storage) & VI (external data sources)	N/A	Batch & stream
(Nadal et al., 2017, 2019)	O (λ-architecture extended with semantic layer)	N/A	Batch & stream
(Vanhove et al., 2015)	O (based on λ-architecture principles, using an enterprise service bus)	N/A	Batch & stream

Since one of the characteristics of Big Data is Velocity meaning that data may income into the system with different speed, both processing modes, batch and stream,

need to be supported in the candidate architecture. Therefore, the processing modes provided in each examined study are added to the table 1. We give the details of the solutions leveraged by studies mentioned in the table 1 in the following subsections to elaborate on the research question RQ1.

3.1 Big Data Warehouse/OLAP Type

3.1.1 Cheetah. The paper (Chen, 2010) presents an original solution of a data warehouse for the analysis of Big Data named Cheetah developed using the MapReduce paradigm. The proposed solution leverages a snowflake schema. The solution includes virtual views that are built on top of such schema by selection of fact table measures and attributes of all related dimensions. User queries are executed on virtual views. The authors also developed a query language that is similar to SQL, allowing to query multiple virtual views and perform aggregation. If a user runs a query on multiple views, all columns included in the query are selected, but if any of the columns is not available in the view, its value is replaced by NULL.

The authors use a data warehouse to analyze advertising data and provide such analysis to customers. It is mentioned in the paper that because of the nature of the data warehouse business environment, schema changes in it occur frequently. The proposed solution supports the evolution of a data warehouse schema in two ways. First, the authors apply slowly changing dimension approach by Ralph Kimbal (Kimball and Ross, 2013). Secondly, changes to the fact tables are implemented through versions, i.e., for each row of a fact table, a version identifier is supplied and information about columns belonging to a particular version is stored in the metadata. The recognition of changes in a data warehouse schema and their treatment is not described in the paper.

The paper also presents an architecture of the system. Data warehouse data is stored in a cluster. Users can use a web interface, command line interface, or Java code with JDBC to execute queries. One of the cluster nodes runs a query engine that uses metadata and converts user queries into MapReduce jobs that are executed on multiple data nodes and calculates the desired result. Each node has a data access primitive (DAP) interface, which is actually a scanner on virtual views.

Physically data warehouse data is stored in a variety of formats: text (CSV), serialized Java object, queue-based binary array and column-based binary array. The authors claim that the last format is preferred since it offers the best query performance, which is proved with experimental evaluation.

3.1.2 An Optimized Distributed OLAP System for Big Data. The authors of the paper (Chen et al., 2017) describe an OLAP system for Big Data analysis as well as the optimization of this system. The system consists of 4 modules:

- In the data acquisition module, source data is obtained: relational database table data is retrieved using batches, and unstructured data (log file data) is obtained as streams.
- All data is stored in the data storage module: the data to be analyzed is stored in a distributed file system, user rights data and metadata are stored in the relational

database, the key-value database stores previously calculated OLAP cubes and their metadata.
- In the OLAP analysis module, user-defined cubes are calculated and an interface for SQL-like queries is provided.
- The main goal of the data visualization module is to provide user connection, user rights management, graphical representation, saving and export of query results, as well as definition of OLAP cubes for the OLAP analysis module.

Various existing tools are used to implement the proposed system architecture. Scoop is used to retrieve data from structured data sources and to load into HDFS, Flume is used to retrieve log file data. Hive is used for the storage of structured source data and Kafka is used for log file data. OLAP analysis is provided by Kylin for pre-calculated cube data and Impala for ad-hoc queries. The authors note that Impala can process original data loaded from data sources, but it works slowly, Kylin works faster but pre-calculates potentially useful cubes. Saiku is used for data visualization.

The authors also present their optimization solutions for the architecture where one relates to automatic metadata generation for the Saiku tool based on Kylin metadata, and the other is related to configuring the cache for the Saiku tool.

Finally, the authors demonstrate an experimental evaluation of their proposed system that shows better performance of Kylin compared to Impala and better performance when queries are cached comparing to the case when they are not cached.

Even though the authors do not mention evolution, the proposed solution can be complemented by evolution support. Its' week point is the lack of integration of data from different data sources.

3.1.3 Modelling and Implementing Big Data Warehouses for Decision Support.

The paper (Santos et al., 2017) is dedicated to the problem of migrating an existing data warehouse based on a traditional star schema to a Big Data warehouse built on Hive. The authors propose laws that allow one to convert the traditional data warehouse model to the Hive data model. The authors suggest that these laws allow to obtain an optimized model and that queries on it run faster.

In the paper, the authors emphasize that originally a traditional data warehouse already exists. The previous data warehouse is also included in a data warehouse architecture that the authors propose for Big Data applications, although authors also mention the architecture modification in case a relational data warehouse did not exist previously in an organization. If a data warehouse already exists, then ETL processes firstly load source data into the existing data warehouse and then, as the second step, they migrate the data from it to the HDFS staging area. The authors use the toolkit Talend Open Studio for Big Data to implement ETL processes. If a data warehouse did not exist before, the first step is omitted. In the third step, the data is modified according to the laws proposed by the authors and loaded into Hive tables. Next, in the fourth step, data from Hive is prepared for analysis with Impala: queries are executed and their results are presented by dashboards and other visualization methods. Users of the system leverage a front-end tool Tableau to analyze data.

The purpose of our paper is to examine architectures that are applicable in any case, whether an organization already has a data warehouse or not, so it is worth analyzing

only a part of the architecture proposed in the paper. In addition to the fact that the architecture used in this paper does not support evolution, it does not analyze and use unstructured data and is only used for batch data loading.

3.1.4 HaoLap. The paper (Song et al., 2015) presents HaoLap - an OLAP system for Big Data implemented in Hadoop. The authors propose a data model utilized in the system and algorithms for execution of aggregate queries, roll-up and drill-down operations and for data distribution among multiple nodes of the system. To store Big Data in the system, the authors utilize an OLAP cube data model with dimension tables structured into hierarchy levels and fact tables which include measures that are distributed over multiple nodes in the overall architecture.

Each query in the system is perceived as an aggregation query with certain conditions on the dimension table values. For each query, the system first finds fact table chunks that contain measures that match the query conditions. Then, measure values corresponding to the conditions set on dimensions are selected from each chunk. Usually, if aggregation is required, then data is grouped for the particular aggregate function based on certain dimension levels. Using dimension surrogate keys and dimension coding and traversing algorithms offered by the authors, firstly measurement values are selected during the Map phase, then the aggregate function is calculated within the group during the Reduce phase and the query result is saved as a chunk in the MapFile. Once all the files are created, the pieces stored in them are combined in the result cube. The authors also describe a data loading process, which transforms original data obtained from data sources into the data model proposed in the paper. Map and Reduce tasks are executed for this purpose.

The solution leverages share-nothing architecture, which consists of the following components:

- Hadoop cluster that stores OLAP cube data that is loaded using the ETL tool;
- A metadata server that contains metadata about dimensions and cubes;
- A job node that manages queries according to the algorithm developed by the authors by first performing a validation of the OLAP service facade query, then configuring the MapReduce task according to the query and finally reporting the query results to the OLAP service facade;
- OLAP service facade, which is a mediator between the OLAP client and the task node and which receives the query, checks for correctness, creates an intermediate form query using the information from the metadata server, provides an intermediate request to the task node, awaits for the result, and supplies it to the OLAP client;
- OLAP client allow users to query data and display cube data.

The authors also perform experimental analysis and compare the performance of the proposed system with other similar solutions: Hive, HadoopDB, Olap4Cloud and HBaseLattice. The authors note that the proposed solution outperforms other available solutions.

3.1.5 Avatara. The paper (Sumbaly et al., 2013) discusses an architecture that applies Big Data technologies and is used for large-scale data processing and OLAP analysis in LinkedIn. The architecture supports evolution of source data by maintaining a schema registry and ensuring that the schema matches the desired structure or is adaptable to it. OLAP functionality is provided by the distributed system Avatara presented in the paper (Wu et al., 2012).

Avatara is used in LinkedIn social network to provide OLAP analysis for each individual user profile and job offer. The specificity of the system is that it contains a large number of small OLAP cubes, because separate cubes are created for each user or job offer and data in each such cube is not large compared to the total size of data.

Avatara consists of two subsystems: offline cube calculation engine and online query processing engine. OLAP cubes are re-calculated every 2 hours, but queries are executed on cubes with a response time of tens of milliseconds. This means that data returned by a query is maximum 2 hours old, but to get a query result one does not have to wait long. Offline engine performs user-defined joins and aggregation from source data, resulting in a generation of multiple small cubes. The offline engine uses Hadoop and generates cube data in a key-value format.

Users leverage an online query engine to define SQL-like queries. Queries are executed by applying such operations as filters, sorting and aggregation on a key-value store. LinkedIn company uses Voldemort to store data. Both engines support cube materialization and developers choose which cubes to materialize.

Cube construction consists of 3 phases. Initially, data is pre-processed using several defined functions, such as data aggregation. In the second phase, dimensions, fact tables and measures are defined. The last step is the cubification, where several small OLAP cubes are generated that are sharded on a chosen key, measure aggregation is also applied. Cube data is stored in MOLAP format. Next, the cubes are loaded into a distributed key-value store. Cube data is supplemented incrementally, therefore, it is possible to update cubes several times a day.

When a query is executed, if the data store supports local storage nodes computation, the query engine supplies query predicates to the store to reduce the amount of data transmitted over the network. A cube with previously calculated and aggregated data is returned from the store.

The authors also mention the evolution problem when a new dimension is added to data. This change is not executed automatically because the structure of cubes must be defined a priori, so in case of a new dimension the developer has to change the cube configuration manually and re-start a cube generation process with a new configuration.

3.1.6 An Iterative Methodology for Big Data Management, Analysis and Visualization. The study presented in the paper (Tardio et al., 2015) proposes a methodology for a construction of a system for Big Data analysis. The system is based on a multidimensional model (MD) and Big Data warehouse. One of the methodology steps is a design of a MD. Information requirements must be defined before the model is constructed. Initially, several data sources are analyzed and a MD is constructed separately for each source. After the creation of several models, they are grouped based on fact

similarity, elements of the models united into each group are compared and the models are integrated. The integration process is carried out several times to get a unified MD.

The authors also discuss a model enrichment process, which may take place when a new data source is added to the system or additional data are discovered by applying data mining methods. To handle changes in Big Data, the authors propose iterative execution of the model design step until such a model is obtained that can satisfy all information requirements. In the context of the Big Data warehouse evolution, the authors' proposed method for model design, integration and enrichment can be used not just for the initial data warehouse construction, but also for evolution when a MD needs to be supplemented due to new information requirements or new data source addition. Problems arise when any of the previously available data sources becomes unavailable.

The authors use Apache Pig and Hive to structure and query source data. After the multidimensional model has been constructed, the authors propose to use different tools to implement a data warehouse, depending on the desired query latency. For high latency, the authors suggest using MapReduce/HDFS-based tools, such as Apache Hive, but for low latency, they propose the use of a database that supports Big Data such as a massive parallel processing database (HP Vertica) or an in-memory approach (Power Pivot and QuickView).

3.2 Data Lake

3.2.1 A Reference Architecture and Model for Sensor Data Warehousing. In the paper (Dobson et al., 2018), the authors propose an architecture and data warehouse model for analyzing sensor data. The architecture is currently in the status of a proposal and has not been implemented in real use cases. The authors point out the implementation and validation of the architecture and model as further work directions. However, since the article is new (published in 2018) and the idea seems perspective, this study was included in our review.

In the proposed architecture, sensor data is retrieved from data sources and loaded into the data lake repository. Both batch and stream modes are supported in this process. Data lake data is periodically copied into the data warehouse using ETL processes. The authors state that all types of data: structured, semi-structured and unstructured, are required for sensor data analysis. From the technology point of view, the authors propose the use of Hadoop and other frameworks supported by it for various tasks, such as stream processing frameworks Apache Storm, Apache Spark, Apache Flink (the latter two could also be used in batch processing mode). HDFS file system is suggested for storage of unstructured data and Apache Hive, SparkSQL or Impala are proposed for structured data. Apache Kylin could be used to support OLAP operations.

The evolution perspective is also mentioned in the paper. The authors believe that data lake could be easily transformed/supplemented to reflect changes in data sources. In such a case ETL processes must be appropriately adapted to take into account changes that have occurred, and evolution must be handled in the data warehouse schema. Even though the authors outline the possible evolution management process, practical methods for it are not proposed in the architecture.

3.2.2 D-Ocean. In the paper (Zhuang et al., 2016) a data management system for unstructured data D-Ocean is described. The architecture of D-Ocean is open and scalable, it provides the possibility to store data in different data stores, has both batch and incremental modes of unstructured data processing, and ensures a search engine for complex query processing. The D-Ocean architecture consists of three layers tools, core and infrastructure.

- There are 7 main modules in the core layer: interfaces, environment controller, repository, analytics component, indexes, and also search and data processing modules. The repository ensures CRUD operations with data. They are afterwards executed by the data processing module that uses tools from other modules, e.g. indexes.
- The infrastructure layer provides the data storage and data processing infrastructure. For the data storage different types of stores can be used including NoSQL databases, for example, HBase, and file systems, for example, HDFS and FastDFS. Other technologies used for the implementation are: Hadoop MapReduce for the data processing, Solr for indexing, and ZooKeeper for coordination of processes.

The complexity of unstructured data is determined by the rich semantics, by other features that can be obtained from raw data, and by links between unstructured data objects, for example nesting and inheritance. To store data according these features D-Ocean provides a model of unstructured data, where the main objects are UObject, Feature, UType, FeatureType and others. D-Ocean supports also an SQL-like query language UQL, that ensures among other options the CRUD type operations, full text search, and aggregation.

The evolution or enhancement of the D-Ocean system is possible in different ways. The data model can be extended using FeatureType and UType objects by defining relevant new types and assigning new processing methods for them. The infrastructure for data storage can be enhanced by defining new interfaces. Also data processing can evolve by adding new plugins into the processor module.

Important role for the consistent operation of the system plays the implementation of the repository. During the execution of user queries the type information and data queries are processed. The meta manager stores the type information into the metastore database, so providing metadata for the functions of the whole system.

Along with the unified data storage system, D-Ocean also provides two modes of data processing: batch and incremental. The processor module sends in the batch mode the data processing tasks directly to Hadoop, but in the incremental mode, a message queue is built to support the real time processing.

3.2.3 Constance. The authors of the paper (Hai et al., 2016) present a data lake system Constance that identifies and manages structural and semantic metadata, integrates and simplifies multiple schemas based on metadata, and offers an interface for query definition and execution.

The system is composed of 3 layers. In the ingestion layer, data from different data sources is loaded into the repository in its original format.

In the maintenance layer, data source metadata is obtained, source schema integration and global schema simplification are conducted. Metadata acquisition from relational sources and partly structured data sources with schema (e.g. XML data with XSD schema) is possible directly from the data source. A structural metadata discovery component is used to obtain metadata for other semi-structured data sources. It identifies indirect metadata using 2 approaches. First, the structural metadata discovery component analyzes data files from the sources and searches for metadata information, such as column names in an Excel table, directory structure, etc. Secondly, relationships between data entities are discovered in source data that are later supplemented by relationships that are often encountered in user queries, i.e. over time, metadata is enriched and updated. The second component included in the maintenance layer is a semantic metadata matching, which performs ontology modeling from metadata, attribute annotation, record linkage, and semantic enrichment. This component constructs a graph representing elements of the source schemas and their relationships. In this way, multiple sources are integrated. After that, schemas are grouped and, as a result, a simplified view of the integrated schema is obtained that filters the most important elements of the schema and their relationships.

The querying layer ensures rewriting and processing of user queries. Queries can be defined by a formal language (JSNOiq) or by leveraging a query formulation component that allows to build a query using key words.

The authors mention that Constance is a flexible system that can connect new data sources to the ingestion layer to supplement the system. Besides, it can be inferred that changes in schemas of data sources would be automatically discovered by the system components in the maintenance layer, which means that data source evolution is naturally supported.

3.3 Virtual Integration

3.3.1 SODIM. The paper (ElSheikh et al., 2013) provides a system SODIM for data integration based on a virtual database approach. The architecture consists of the following components:

- Data from data sources is obtained through web services. Information about each web service is available in WSDL format. By executing a user query, the architecture determines services required for the response and dynamically employs them using service descriptions (WSDL) and DAIOS framework.
- A user exploits the integrated service to define a query. The integrated service converts the user query into a job that executes a partial query on each required data source and combines data from different data sources.
- Hadoop is used for the job execution. It divides the job between workers and combines individual results.

Experiments with different hardware configurations are also performed and described in the paper. Unfortunately, no detailed implementation of the architecture is discussed, for example, it is not mentioned how jobs are generated and how user queries are transformed into partial queries.

3.3.2 VDB-MR. The authors of the paper (Yuan et al., 2010) propose the use of MapReduce technology to query the virtual database (VDB-MR). VDB is a virtual integration technology that allows users to define a global schema that combines several different source schemas. Source data is not transferred physically to the global system. Traditional VDB architecture consists of 4 components:

- Mapper defines a global integrated schema and mapping of its elements to source schemas,
- Publisher manages user queries, distributes them to execute on different data sources,
- Executer performs queries on data sources, combines query results, solves conflicts,
- Wrapper operates on a separate source, converts queries to a format that is understandable by a specific source, and converts query results obtained from that source.

The authors of the paper claim that the traditional VDB architecture cannot execute queries on large amounts of data and combine query results within a reasonable time, so the authors suggest using MapReduce in the query execution step.

The architecture proposed in the paper supports 4 operations. Map operation defines the mapping of source schema elements to the global schema. Select operation selects data (similar to a simple SQL Select query). Join operation connects multiple tables. Update operatoion performs data modifications (delete, update, insert).

The process of performing each operation using MapReduce approach is described in the paper. For example, the Select operation is executed as follows. The Manage process is executed. It divides a query into subqueries where each subquery is executed on a separate data source. After that, the Map process is executed when a worker runs a query on a particular data source. As a result, data in a format of key:value is generated where the key consists of the primary key of a table from the global schema and the value is a set of values of the other attributes obtained from the particular data source. Next, the Reduce process is executed when the reduce worker converts data obtained from map workers. Data records with the same key obtained from different data sources are merged and their attributes are combined. If the same attributes with different values are obtained from different data sources, the first value of each such attribute is selected. Finally, the Collect process is executed. It converts the data obtained from the Reduce process into a user-friendly format.

The authors of the paper have also conducted an experimental evaluation of their proposed VDB implementation, which showed the better performance compared to another VDB system (without the use of MapReduce) and parallel RDBMS.

The proposed system does not support the schema evolution directly, but it can be implemented manually, for example, by mapping redefinition. Similarly, new data sources can be added. Data maintenance after evolution is not necessary as up-to-date data is obtained directly from sources.

3.4 Polystore

3.4.1 BigDAWG. According to the definition given in (Gadepally et al., 2017) *"A polystore system is any database management system (DBMS) that is built on top of multiple, heterogeneous, integrated storage engines."* BigDAWG (Gadepally et al.,

2017) is one of the possible implementations of a polystore. The base layer is formed of a number of data storage engines. These engines are organized in multiple islands, that each have a common data model and a set of operations

The islands component is connected with one or more storage engines by means of shim components. The main goal of a shim is to transform the operations of an island into the query language of a particular storage engine. BigDAWG also includes cast components, which if necessary can transfer data from one data storage engine to another.

The main part of BigDAWG is a middleware, which is an interface to the storage engines and uses the islands. The middleware consists of the following components:

- Optimizer transforms the original query into many possible query plans of each subquery for potential engines;
- Monitor uses the performance data of previous query execution to find out the best combination of query plan and execution engine;
- Executor chose the best way how to join the data collections and then executes the query;
- Migrator moves the data from one engine to another, if the execution plan predicts that it will be more efficient.

BigDAWG stores the necessary metadata in a metadata catalogue. For example, the migrator and executor use data about systems data storage engines: their location, data objects and others. For the implementation of the metadata catalogue, the PostgreSQL database is used and the following tables are stored: Engines (engine names, login information), Databases (names, relation to particular engine, login information), Objects (data objects, e.g. tables, field names, relation to a particular database), Shims (possibility to integrate a shim with a particular island), Casts (possible moves between engines).

The initial BigDAWG implementation supports three open source database engines: PostgreSQL, Apache Accumulo (NoSQL), and SciDB (NewSQL), and accordingly relational, array and text islands. The middleware uses Docker containers.

3.4.2 Myria. Myria (Wang et al., 2017) is a system for integrated data management and analysis, that have a special execution engine MyriaX. At the same time execution plans for other different execution engines can be generated, including Spark, SciDB and PostgreSQL.

Each of the Big Data processing systems that are integrated into the Myria polystore supports more efficiently a particular type of tasks, for example, SciDB supports array processing, but MyriaX executes efficiently iterative queries. These systems often are deployed in the cloud, so the Myria user gets the advantage of each particular system without caring about usage of them separately.

The query execution engine MyriaX mainly is a traditional parallel database system extended with different features, for example, usage of data from different sources: HDFS, Amazon cloud and others. Myria is service oriented and developed as a cloud service. The main application area is the relational data.

Myria is designed for analysis of data that is stored in different back-end systems covering also those with non-relational data models. Myria provides common services for the whole polystore environment including query execution and optimization. For the query execution in federated environment and for the optimization, Relational Algebra Compiler RACO is developed. RACO is based on the relational algebra, but also supports the compilation of relational expressions for the array, graph or key-value engines. The Myria logical algebra expressions are translated into physical expressions of one or more supported back-end systems. The execution plans are also generated, which also include data movements among back-end systems.

The users can express their queries using MyriaL, a relational query language with extensions. MyriaL is similar to the procedural language PL/SQL. Pyton also can be used through API or user defined functions.

3.5 Variations of Standard Types

3.5.1 AsterixDB. The paper (Alsubaiee et al., 2014) discusses AsterixDB Big Data management system (BDMS) that allows to store semi-structured data in partitions and execute queries on them. Data are stored in a Dataverse, which is similar to a database concept in traditional relational databases. Schema of the data to be stored is defined by Datatypes that may be closed or open, meaning that open datatypes allow additional data properties that are not included in the schema specification but require all data properties defined in the schema. It is possible to load data into the system or execute queries on external data without storing them locally. This way virtual integration may be implemented using the proposed BDMS. Data feeds are designed to load data. They can load data both in batch or stream modes. AsterixDB supports various types of queries, user defined functions, primary and secondary indexes for fast querying, data insertion and deletion operations.

Data storage mechanism implemented in AsterixDB allows schema evolution by allowing storage of new data that does not conform to the scheme, but in case of a new data source, a new data type must be defined at least minimally. If any data field described as mandatory in the data type is removed from the source data, the system will not allow such data to be loaded from that data source. Data maintenance is supported by means of data feeds, data insertion and deletion operations.

3.5.2 Bolster. The paper (Nadal et al., 2017) presents a software reference architecture for Big Data processing and analysis. The proposed architecture solves various kinds of problems related to Big Data characteristics, including Variability or evolution problems. The architecture is based on λ-architecture and is extended with semantic layer components. The proposed architecture is composed of 4 layers:

- In the batch layer massive volumes of data are periodically obtained from batch data sources and loaded into a data lake in their original format. Then, data from the data lake is processed with iterative algorithms to meet requirements of the particular system and moved to the serving layer.

- In the speed layer stream data is put through a dispatcher component that validates data quality and routes data either to the data lake for subsequent batch processing or to the stream processing component for real-time analysis.
- Processed data from both the batch layer and speed layer is available for analysis by end users in the serving layer as batch views and real-time views via the query engine component. Furthermore, it is also possible to leverage such views as data sources for additional transformations.
- The operation and decisions made by various components of the architecture, such as batch processing component, dispatcher, stream processing component and query engine, are regulated by the metadata stored in the form of RDF ontologies in the metadata repository and maintained by the metadata management system in the semantic layer of the architecture.

A solution to handling data source evolution problems in the architecture is presented in the paper (Nadal et al., 2019). The proposed approach could be used when source data is obtained using wrappers, such as Rest API in JSON format, and loaded into a Big Data ecosystem. The authors use a local-as-view (LAV) approach to define a mapping between elements of a data source and elements of a target or global schema. LAV means that the source elements are defined using the elements of the global schema. This approach makes it easy to process changes in data sources without changing analytic queries on a global schema.

The authors propose the use of Big Data integration ontology to define the integrated schema, source schemas, and their versions, and LAV mappings between them as a graph. The ontology is also used to specify queries. When a change at a data source occurs, the system steward supplements the ontology with a new release. A new wrapper with unchanged and new attributes is assigned to the changed data source, and a mapping between new attributes and a target schema is defined. The proposed approach is usable only if the source data has a schema and cannot be directly applied for unstructured data. The paper does not discuss the determination of change occurance at data sources and the evolution of the global schema has not been considered.

In the paper (Nadal et al., 2017) the authors also name available tools and technologies that could be used to instantiate components of Bolster. In the batch layer ad-hoc scripts or existing drivers such as Apache Sqoop may be applied for batch data ingestion; HDFS or Amazon S3 may be leveraged to implement batch data storage in a data lake, and Apache MapReduce, Amazon Elastic MapReduce, Apache Spark or Apache Flink may be used for batch processing.

To implement stream ingestion in the speed layer, the authors recommend the use of tools for message queue processing, such as Apache ActiveMQ and RabbitMQ, as well as tools specially aimed at stream data acquisition, such as Apache Kafka and AWS Kinesis Firehose. For stream data routing, such tools as Apache Flume or Amazon Kinesis Streams are applicable. Apache Spark Streaming, Apache Flink Streaming and Amazon Kinesis Analytics may be used to process stream data and generate corresponding real-time views.

Many distributed systems are suitable to store batch view data. Several potential examples mentioned in the paper include Oracle, Postgres-XL, MySQL Cluster, Apache HBase, Apache Cassandra, Amazon DynamoDB, Voldemort, Amazon Redshift, Apache

Kudu, Neo4j, OrientDB, MongoDB, RethinkDB, SAP Hana, NuoDB, VoltDB. Existing tools that may be used to instantiate real-time views are also named. These are in-memory storage systems, such as Redis, Elastic or Amazon ElastiCache, and a database that runs queries on streams PipelineDB. A query engine may be represented as Apache Kylin, Kibana or Tableau.

Finally, in the semantic layer Apache Stanbol, Apache Atlas, Cloudera Navigator or Palantir may be used as metadata management systems, and Virtuoso or graph databases may be chosen for metadata storage.

3.5.3 Tengu. Tengu (Vanhove et al., 2015) is an experimentation platform for data analysis. The platform supports an automatic installation of its components according to the needs of a particular project. This experimentation platform is compatible with GENI (US federation of testbeds) and Fed4FIRE (EU federation of testbeds). The automatic installation includes different Big Data analysis frameworks, relational databases and cloud services.

Tengu architecture consists of three major parts: a computational unit, data store and a set of resources for a particular application. The communication among these parts is supported by the Enterprise Service Bus that acts as a middleware. The computational unit is based on the principles of the λ-architecture that supports batch and stream data analysis. The batch layer provides a view over the whole data, but the new data are arriving in the speed layer. The new data are analysed and afterwards added also to the batch layer. These data are then used in the next analysis step and also deleted from the speed layer to avoid data redundancy. For storage of data, Tengu offers a variety of options to provide an optimal solution for a specific application; multiple simultaneous data stores are also possible. TENGU supports also the data store transformations which means that projects data store can be replaced with another data store if necessary.

Tengu platform provides evolution support, offering the option of automatic transformation from one storage type to another. The necessity for the evolution is caused by the usage of an experimental application over a time. The transformation is performed by the already mentioned Lambda architecture. The transformation starts with the batch layer where the schema and data of the original data stores snapshot are transformed according to the requirements of the new data store. At the same time the speed layer transforms the new queries. The next step is the creation of a new data store using the transformed snapshot and transformed queries from the speed layer. Then the platform switches the application from the old data store to the new one. The Enterprise service bus provides the communication between the application and the data store and also provides the query transformation. The user can continue to use a convenient query language in a new data store.

Tengu supports Hadoop and Storm for batch and speed layers, MySQL, Cassandra and ElasticSearch for the data storage. Tengu supports usage of OpenStack, Tomcat, Zookeeper and Kafka, but after minor changes the architecture allows to integrate also other new technologies.

4 Analysis of Architecture Characteristics

4.1 Evolution

Because of the nature of Big Data, evolution problems are more topical than in traditional structured data storage and processing systems. The table 2 summarizes approaches to support evolution applied in the studied architectures to answer our third research question RQ3.

In our research, we classified evolution problems that must be solved into three categories. Data maintenance must be provided by the architecture, since Big Data are dynamic and up-to-date data from data sources are necessary for the analysis. Data required for the analysis are often semi-structured or even unstructured, it is important to dicover changes in such data and to handle them properly. New data sources may become available and necessary to support new or more valuable analysis capabilities provided by the Big Data analysis system. If Big Data available for the analysis are structured into a schema (integrated target or global schema), evolution of such schema (denoted as schema evolution in Table 2) may occur because of changes in information requirements of the system, for example, when additional data may become necessary for decision-making, or after changes in data sources.

Evolution is handled in different ways in the studied architectures. Data maintenance is supported by ETL processes in Big Data warehousing architectures. Data lakes (Dobson et al., 2018; Hai et al., 2016; Zhuang et al., 2016) accumulate source data in its' original format. Virtual integration architectures (ElSheikh et al., 2013; Yuan et al., 2010) do not store copies of source data, therefore, maintenance problem is solved naturally. Polystores (Wang et al., 2017), λ-architectures (Nadal et al., 2017) and other architecture types (Alsubaiee et al., 2014) apply specific operations, such as union, insert and delete or do not consider a data maintenance problem at all.

The support of inclusion of new data sources is very limited in the studied architectures. Mostly it must be handled manually or semi-automatically. The only study capable of automatic data source discovery is a data lake (Hai et al., 2016) where new data source metadata are detected by the architecture components.

Schema evolution is not considered in the majority of Big Data warehouse architectures or is handled by slowly changing dimensions and versioning approaches that are well known in the field of traditional relational data warehouses (Chen, 2010). Architectures that include data lakes (Hai et al., 2016; Zhuang et al., 2016) use metadata to process changes in the integrated target schemata. In case of other architecture types, schema evolution is not considered or may be implemented only manually.

Table 2: Evolution Support

Paper	Data Maintenance	New Data Sources	Schema Evolution
(Chen, 2010)	User-defined ETL	N/A	Slowly changing dimensions, fact table schema versions

Paper	Data Maintenance	New Data Sources	Schema Evolution
(Chen et al., 2017)	Implemented in data acquisition module	N/A	N/A
(Santos et al., 2017)	User-defined ETL	N/A	N/A
(Song et al., 2015)	User-defined ETL	N/A	N/A
(Sumbaly et al., 2013; Wu et al., 2012)	Cube data is bulk loaded	N/A	New dimensions are added manually and require cube recomputation
(Tardio et al., 2015)	Storage is supplemented with new data	Iterative model enrichment	Iterative model enrichment
(Dobson et al., 2018)	User-defined ETL	N/A	N/A
(Zhuang et al., 2016)	N/A	N/A	Data model is provided for unstructured data, two extensible types allow addition of new features
(Hai et al., 2016)	Data is loaded as-is in the ingestion layer	New source metadata are discovered automatically	Metadata of updated schema are obtained
(ElSheikh et al., 2013)	Naturally supported	Yes (new web services may be added dynamically)	Yes (web service description has to be updated dynamically)
(Yuan et al., 2010)	Naturally supported	N/A	N/A
(Gadepally et al., 2017)	N/A	Additional islands or database engines can be supported	N/A
(Wang et al., 2017)	New data for base relations are added with union	Additional database engines can be supported	N/A
(Alsubaiee et al., 2014)	Inserts and deletes	Schema needs to be defined	Partially
(Nadal et al., 2017, 2019)	Provided by batch and speed layers	A new release is added to the ontology	A new release is added to the ontology
(Vanhove et al., 2015)	N/A	New data sources are included automatically by Enterprise service bus using live transformation between two data stores	N/A

4.2 Metadata

To answer our forth research question RQ4, we analyzed types and discovery methods of metadata applied in each architecture included in this study. The summary of metadata utilization is given in the table 3. One of the characteristic of Big Data is variety, therefore, we also classified the studied architectures with respect to supported data formats: S-Structured, SS-Semi-structured, US-Unstructured. We also considered metadata modelling approach if applicable in the analyzed architectures, such as SOR- schema-on-read or SOW- schema-on-write, and data provenance issue.

Table 3: Metadata

Paper	Variety	Types of Metadata	Metadata Discovery Method	Modelling Approach	Data Provenance
(Chen, 2010)	S, SS	Structural metadata (DW schema, fact table schema versions)	N/A	N/A, DW schema is predefined	None
(Chen et al., 2017)	S, SS	Structural metadata (table structure, cube metadata)	Metadata for data visualization tool is obtained automatically from OLAP engine metadata	N/A, DW schema is predefined	None
(Santos et al., 2017)	S	N/A	N/A	N/A, DW schema is predefined	None
(Song et al., 2015)	S	Structural metadata (MD schema)	N/A	N/A, DW schema is predefined	None
(Sumbaly et al., 2013; Wu et al., 2012)	S	N/A	N/A	N/A, cube schema is predefined	None
(Tardio et al., 2015)	S, SS, US	Structural metadata (MD model)	N/A	N/A, DW schema is predefined	None
(Dobson et al., 2018)	S, SS, US	N/A	N/A	N/A, DW schema is predefined	None
(Zhuang et al., 2016)	US	Metastore contains type information	The meta manager is used	SOR	None

Paper	Variety	Types of Metadata	Metadata Discovery Method	Modelling Approach	Data Provenance
(Hai et al., 2016)	S, SS, US	Structural metadata, semantic metadata	Structural metadata discovery, Semantic metadata matching	SOR	Yes
(ElSheikh et al., 2013)	S, SS	Structural metadata (WSDL)	WSDL has to be provided by a web service	SOR	Yes
(Yuan et al., 2010)	S, SS	Structural metadata (global and data source schemata, mappings)	N/A	N/A, global schema is predefined	Mappings between source schemata and global schema
(Gadepally et al., 2017)	S, SS, US	Metadata Catalog contains data about engines, databases, objects, shims, casts	N/A	SOR, SOW	None
(Wang et al., 2017)	S, SS	N/A	N/A	SOW	None
(Alsubaiee et al., 2014)	S, SS	N/A	N/A	SOR, SOW	None
(Nadal et al., 2017, 2019)	S, SS, US	Domain vocabulary, source and global schemata, mappings	Data Source Register discovers schemata of all new data sources	SOR	Data transformations are registered in logs
(Vanhove et al., 2015)	S, SS, US	Enterprise service bus is provided to coordinate usage of batch and speed layer	Algorithms are implemented to analyze batch data sets, a batch view is created	Batch view and Speed view are provided	None

Our results show that structural metadata, such as data source or global schema metadata or mappings, are usually utilized in the majority of examined approaches and more advanced studies (Hai et al., 2016; Nadal et al., 2017) provide also semantic metadata. Metadata discovery is implemented in the studies that are based on the data lake

architecture (Hai et al., 2016; Zhuang et al., 2016) or employ data lake as one of the components (Nadal et al., 2017).

As to the metadata modelling approaches, a conclusion can be made that Big Data warehouse architectures require manual metadata definition, data lake architectures follow schema-on-read metadata aquisition method, but polystores either support schema-on-write approach or both mentioned approaches. Although data provenance is an important feature that can be used not just to ensure data quality, but also to handle evolution at data sources, no mechanisms to provide it are proposed in the majority of the reviewed studies. Just the studies based on the data lake (Hai et al., 2016; Nadal et al., 2017) or virtual integration architecture (ElSheikh et al., 2013; Yuan et al., 2010) offer such solutions.

5 Big Data Warehouse Evolution Architecture

To solve the Big Data evolution problem, we propose an architecture that allows to store and process structured and unstructured data at different levels of detail, analyze them using OLAP capabilities and semi-automatically manage changes in requirements and data expansion. The idea of the Big Data warehouse architecture was inspired by our previous work on traditional data warehouse evolution framework (Solodovnikova, 2007). We extended the data warehouse evolution framework to the context of Big Data.

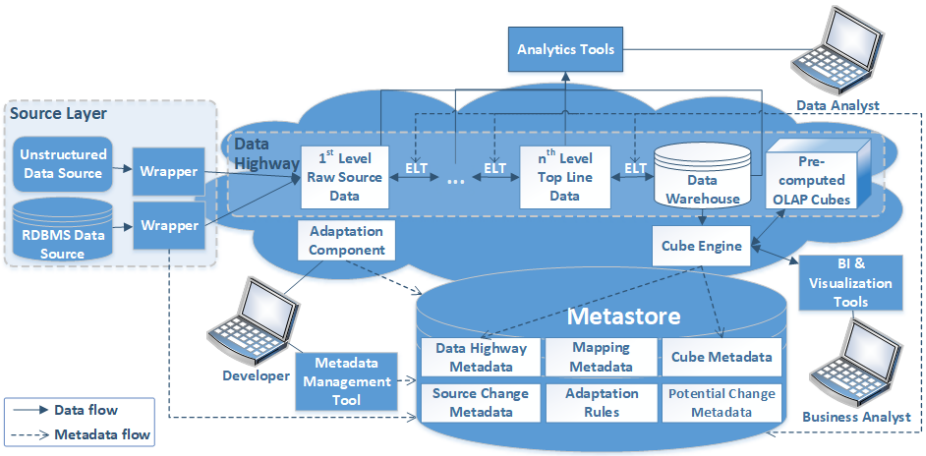

Fig. 1: Big Data warehouse architecture

5.1 Architecture Components

The proposed architecture consists of several components (Figure 1). In the source layer, wrappers obtain structured and unstructured data from data sources and load them

into the system at different rates in their original format. We adopt the idea proposed by (Kimball and Ross, 2013) to build a highway of data at different levels of latency. Starting from the raw source data, each next level data are obtained from the previous level data and are updated less often. Besides, the latter level data from multiple heterogeneous sources are integrated and aggregated and finally are transformed into a structured data warehouse schema.

Since data in the proposed architecture are firstly copied in their original format and transformed at the later stage, ELT (Extract, Load, Transform) processes are responsible for data preprocessing. To gain structured data from unstructured sources, advanced methods such as data mining or sentiment analysis must be performed. After being loaded into the data warehouse, the data at each level of the highway are supplemented by the generated surrogate keys of dimensions to ensure data provenance and enable handling of evolution. Since ELT processes augment data at the lower level by information obtained during the processing and transformations performed at the higher level, it is possible to join data from different levels of the data highway to perform a more valuable analysis.

The adaptation component is responsible for handling changes in data sources. The main idea is to generate several potential changes in a data warehouse or other levels of the data highway for each change in a data source and to allow a developer to choose the most appropriate change that must be implemented. To implement certain kinds of changes, the developer needs to supply via the adaptation component additional data that cannot be identified automatically.

We plan to support different kinds of analysis. OLAP cubes may be explored by business analysts in the form of dashboards, charts or other reports and by performing OLAP operations using business intelligence and visualization tools. Since the volume of data stored in the data warehouse may be too large to provide a reasonable performance of data analysis queries, the cube engine component precomputes various dimensional combinations and aggregated measures for them. Apart from OLAP operations, data analysts may apply advanced analysis techniques (for example, data mining) by means of utilizing existing analytics tools or implementing ad-hoc procedures.

5.2 Metadata Management

One of the most essential components of the proposed architecture is the metastore that incorporates six types of interconnected metadata necessary for the operation of other components of the architecture.

- Data highway metadata describe semantics and schemata of Big Data stored in the different levels of the system.
- Cube metadata describe schemata of precomputed cubes and are leveraged not only during the cube computation process but also for execution of queries.
- Mapping metadata define the logics of ELT processes. They store the correspondences between data obtained from the sources and data items of the data highway.
- Information about changes in data sources is accumulated in the source change metadata. Such information may be obtained from wrappers or during the execution of ELT processes. We defined a set of atomic changes that may happen to data

highway levels and for each change, we determined metadata that must be recorded to describe the change in the metastore.
- Adaptation rules specify adaptation options that must be implemented for different types of changes.
- Finally, potential change metadata accumulate proposed changes in the data warehouse schema.

The metadata models that describe schemata of the data highway, source data and changes of the proposed metadata are presented in detail in (Solodovnikova et al., 2019).

To maintain the information in the metastore, a developer utilizes the metadata management tool. In addition, the metadata management tool allows the developer to initiate changes in the data highway and ELT procedures to handle new or changed requirements for data. The history of chosen changes that are implemented to propagate evolution of data sources, as well as changes performed directly via the metadata management tool, are also maintained in the potential change metadata.

5.3 Implementation

As a proof of concept, we have applied our proposed approach to the publications big data system. The system integrates data about publications authored by the faculty and students of the University of Latvia from one structured and three semi-structured data sources and provides the unified data for analysis in the data warehouse. The data highway of the system is composed of three levels. The first level contains original data incoming from data sources. We use Scoop to extract and transfer data from the relational database into Hive tables. Data from other sources obtained in XML format are first pulled from the API and saved in Linux file system and then data are transferred into HDFS using a script with HDFS commands. At the second level, semi-structured data are transformed into structured Hive tables. Data at this level are not yet fully integrated. Finally, the third level of the highway is a data warehouse implemented in Hive.

We implemented the metastore as a relational database (Oracle) and gathered automatically schema metadata from the structured data source since we had the full access to it. To process data extracted from external sources in XML format, we created a procedure that analyses the structure of XML documents and other properties and generates necessary metadata in the metastore. The procedure is also able to compare metadata existing in the metastore with the structure of actual XML files and detect changes that are registered in the source change metadata.

We developed a prototype of the metadata management tool that currently displays the metadata gathered in the metastore, allows a developer to track all changes discovered in data automatically and add information about changes manually. During the operation of the publication big data system, multiple changes were discovered in data sources and other data highway levels by comparing existing metadata and data sets extracted from data sources. We verified the information gathered about changes in the metadata management tool.

6 Conclusions and Future Work

Based on the analysis of existing architectures for Big Data processing, a conclusion can be made that none of the solutions fully support all evolution problems, i.e. evolution of data sources and changes in information requirements. The review papers as well as some papers on data warehouse for Big Data applications agree on the existence of evolution problems in the context of Big Data and the lack of solutions to these problems. In this paper, we have proposed a data warehouse architecture that will be able to overcome the disadvantages of other solutions. The architecture will provide OLAP analysis (including ad-hoc queries) of both structured and unstructured Big Data that are loaded from multiple data sources into the system at different speeds and automatically or semi-automatically process changes to both data sources and user requirements.

The architecture has not been fully implemented, therefore, our future research directions include a development of algorithms for automatic and semi-automatic change treatment. For the implementation of the Big Data warehouse architecture, we intend to utilize the existing tools and technologies as well as to implement the original solutions. After the implementation of the change handling methods, we plan to evaluate our approach experimentally by applying TPC-H benchmark.

Acknowledgments

This work has been partly supported by the European Regional Development Fund (ERDF) project No. 1.1.1.2./VIAA/1/16/057.

References

Abaker I., Hashem T., Yaqoob I., Anuar N.B., Mokhtar S., Gani A., Khan S.U. (2014). The rise of big data on cloud computing: Review and open research issues, *Information Systems*, 47, C, 215, 98-115.

Ahmed W., Zimnyi E., Wrembel R. (2014). A Logical Model for Multiversion Data Warehouses, *Proceedings of 16th International Conference on Data Warehousing and Knowledge Discovery*, Munich, Germany, 2014, 23-34.

Alsubaiee S., Altowim Y., Altwaijry H., Behm A., Borkar, V., Bu Y., Carey M., Cetindil I., Cheelangi M., Faraaz K., Gabrielova E., Grover R., Heilbron Z., Kim Y., Li C., Li G., Ok J.M., Onose N., Pirzadeh P., Tsotras V., Vernica R., Wen J., Westmann T. (2014). AsterixDB: A scalable, open source BDMS, *Proceedings of the VLDB Endowment*, 7, 14, 1905-1916.

Bentayeb F., Favre C., Boussaid O. (2008). A User-driven Data Warehouse Evolution Approach for Concurrent Personalized Analysis Needs, *Integrated Computer-Aided Engineering*, 15, 1, 21-36.

Ceravolo P., Azzini A., Angelini M., Catarci T., Cudr-Mauroux P., Damiani E., Mazak A., Van Keulen M., Jarrar M., Santucci G., Sattler K., Scannapieco M., Wimmer M., Wrembel R., Zaraket F. (2018). Big Data Semantics, *Journal Data Semantics*, 7, 2, 65-85.

Chen S. (2010). Cheetah: A High Performance, Custom Data Warehouse on Top of MapReduce, *VLDB Endowment*, 3, 2, 1459-1468.

Chen W., Wang H., Zhang X. (2017). An optimized distributed OLAP system for big data, *Proceedings of the 2nd IEEE International Conference on Computational Intelligence and Applications*, Beijing, China, 2017, 36-40.

Cuzzocrea A., Bellatreche L., Song I. (2013). Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions, *Proceedings of 16th international workshop on Data warehousing and OLAP*, San Francisco, CA, USA, 2013, 67-70.

Dobson S., Golfarelli M., Graziani S., Rizzi S. (2018). A Reference Architecture and Model for Sensor Data Warehousing, *IEEE Sensors Journal*, 18, 7659-7670.

Duggan J., Elmore A.J., Stonebraker M., Balazinska M., Howe B., Kepner J., Madden S., Maier D., Mattson T., Zdonik S. (2015). The BigDAWG Polystore System, *SIGMOD Rec.*, 44, 2, 11-16.

ElSheikh G., ElNainay M.Y., ElShehaby S., Abougabal M.S. (2013). SODIM: Service Oriented Data Integration based on MapReduce, *Alexandria Engineering Journal*, 52, 313-318.

Hai R., Geisler S., Quix C. (2016). Constance: An Intelligent Data Lake System, *Proceedings of the 2016 International Conference on Management of Data*, San Francisco, California, USA, 2016, 2097-2100.

Holubov I., Svoboda M., Lu J. (2019) Unified Management of Multi-model Data, *Conceptual Modeling. ER 2019. Lecture Notes in Computer Science*, 11788.

Gadepally V., O'Brien K., Dziedzic A., Elmore A., Kepner J., Madden S., Mattson T., Rogers J., She Z., Stonebraker M. (2017). BigDAWG version 0.1, *Proceedings of IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA, 2017, 1-7.

Golfarelli M., Lechtenbrger J., Rizzi S., Vossen G. (2006). Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation, *Data & Knowledge Engineering*, 59, 2, 435-459.

Kaisler S., Armour F., Espinosa J.A., Money W. (2013). Big Data: Issues and Challenges Moving Forward. *Proceedings of 46th Hawaii International Conference on System Sciences*, Wailea, Maui, HI, USA, 2013, 995-1004.

Kimball R., Ross M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd edition, John Wiley & Sons, Inc.

Malinowski E., Zimnyi E. (2008). A Conceptual Model for Temporal Data Warehouses and Its Transformation to the ER and the Object-Relational Models, *Data & Knowledge Engineering*, 64, 1, 101-133.

Marz N., Warren J. (2013). *Big Data: Principles and best practices of scalable realtime data systems*, Manning Publications.

Nadal S., Herrero V., Romero O., Abell A., Franch X., Vansummeren S., Valerio D. (2017). A software reference architecture for semantic-aware Big Data systems, *Information and Software Technology*, 90, 75-92.

Nadal S., Romero O., Abell A., Vassiliadis P., Vansummeren S. (2019). An integration-oriented ontology to govern evolution in Big Data ecosystems, *Information Systems*, 79, 3-19.

Santos M.Y., Martinho B., Costa C. (2017). Modelling and implementing big data warehouses for decision support, *Journal of Management Analytics*, 4, 2, 111-129.

Shvachko K., Kuang H., Radia S., Chansler R. (2010). The Hadoop distributed file system, *MSST2010, IEEE 26th Symposium on Mass Storage Systems and Technologies*, Washington, DC, USA, 2010, 110.

Solodovnikova D. (2017). Data Warehouse Evolution Framework, *Proceedings of Spring Young Researcher's Colloquium on Database and Information Systems*, Moscow, Russia, 2007, 4.

Solodovnikova D., Niedrite L., Kozmina N. (2015). Handling evolving data warehouse requirements, *New Trends in Databases and Information Systems, Proceedings of ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD*, Poitiers, France, 2015, 334-345.

Solodovnikova D., Niedrite L., Niedritis A. (2019). On Metadata Support for Integrating Evolving Heterogeneous Data Sources, *New Trends in Databases and Information Systems, Proceedings of ADBIS 2019 Short Papers and Workshops*, Bled, Slovenia, 2019, 378-390.

Song J., Guo C., Wang Z., Zhang Y., Yu G., Pierson J. (2015). HaoLap: A Hadoop based OLAP system for big data, *Journal of Systems and Software*, 102, 167-181.

Stefanowski J., Krawiec K., Wrembel R. (2017). Exploring Complex and Big Data, *International Journal of Applied Mathematics and Computer Science*, 27, 4, 669-679.

Sumbaly R., Kreps J., Shah S. (2013). The Big Data Ecosystem at LinkedIn, *Proceedings of ACM SIGMOD International Conference on Management of Data*, New York, New York, USA, 2013, 1125-1134.

Tardio R., Mate A., Trujillo J. (2015). An Iterative Methodology for Big Data Management, Analysis and Visualization, *Proceedings of the 2015 IEEE International Conference on Big Data*, Santa Clara, California, USA, 2015, 545-550.

Thakur G., Gosain A. (2011). DWEVOLVE: a Requirement Based Framework for Data Warehouse Evolution, *ACM SIGSOFT Software Engineering Notes*, 36, 6, 1-8.

Thenmozhi M., Vivekanandan K. (2014). An Ontological Approach to Handle Multidimensional Schema Evolution for Data Warehouse, *International Journal of Database Management Systems*, 6, 3, 33-52.

Vanhove T., Van Seghbroeck G., Wauters T., De Turck F., Vermeulen B., Demeester P. (2015). Tengu: An experimentation platform for big data applications, *Proceedings of IEEE 35th International Conference on Distributed Computing Systems Workshops*, Columbus, OH, USA, 2015, 42-47.

Wang J., Baker T., Balazinska M., Halperin D., Haynes B., Howe B., Hutchison D., Jain S., Maas R., Mehta P., Moritz D., Myers B., Ortiz J., Suciu D., Whitaker A., Xu S. (2017). The Myria Big Data Management and Analytics System and Cloud Services, *CIDR*.

Wojciechowski A. (2018). ETL workflow reparation by means of case-based reasoning, *Information Systems Frontiers, 20*, 1, 21-43.

Wu L., Sumbaly R., Riccomini C., Koo G., Kim H.J., Kreps J., Shah S. (2012). Avatara: OLAP for Web-scale Analytics Products, *VLDB Endowment*, 5, 12, 1874-1877.

Yuan Y., Wu Y., Feng X., Li J., Yang G., Zheng W. (2010). VDB-MR: MapReduce-based distributed data integration using virtual database, *Future Generation Computer Systems*, 26, 8, 1418-1425.

Zhuang Y., Wang Y., Shao J., Chen L., Lu W., Sun J., Wei B., Wu J. (2016). D-Ocean: an unstructured data management system for data ocean environment, *Frontiers of Computer Science*, 10, 2, 353-369.

Received November 4, 2019 , revised January 10, 2020, accepted February 4, 2020