Big Data Research 2 (2015) 166–186

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems

Pekka Pääkkönen *,[1], Daniel Pakkala [1]

VTT Technical Research Centre of Finland, Kaitoväylä 1, 90570, Oulu, Finland

ARTICLE INFO

Article history:
Received 22 April 2014
Received in revised form 18 December 2014
Accepted 11 January 2015
Available online 2 February 2015

Keywords:
Big data
Reference architecture
Classification
Literature survey

ABSTRACT

Many business cases exploiting big data have been realised in recent years; Twitter, LinkedIn, and Facebook are examples of companies in the social networking domain. Other big data use cases have focused on capturing of value from streaming of movies (Netflix), monitoring of network traffic, or improvement of processes in the manufacturing industry. Also, implementation architectures of the use cases have been published. However, conceptual work integrating the approaches into one coherent reference architecture has been limited. The contribution of this paper is technology independent reference architecture for big data systems, which is based on analysis of published implementation architectures of big data use cases. An additional contribution is classification of related implementation technologies and products/services, which is based on analysis of the published use cases and survey of related work. The reference architecture and associated classification are aimed for facilitating architecture design and selection of technologies or commercial solutions, when constructing big data systems.

1. Introduction

Many big data use cases have been realised, which create additional value for companies, end users and third parties. Currently, real time data is gathered from millions of end users via popular social networking services. For example, LinkedIn [1] collects data from users, and offers services such as “People you may know”, skill endorsements or news feed updates to end users based on analysis of the data. Another example is Netflix, which uses big data for providing recommendations and ranking related services to customers [2]. Twitter uses collected data for real time query suggestion and spelling corrections of their search algorithm [3]. Analysis of collected data also increases understanding of consumers, which is an important asset for the big data companies. Value from data can also be extracted with other applications such as monitoring of network traffic [4] or improving manufacturing process of digital displays [5].

A wide variety of technologies and heterogeneous architectures have been applied in the implementation of the big data use cases. The publications have mainly concentrated on describing architectures of individual contributions by large big data companies such as Facebook [6] or LinkedIn [1]. On the other hand, architectural work combining the individual reports into one coherent reference architecture has been limited, although the first contributions have been made [7–10]. Technology independent reference architecture and categorization of related implementation technologies and services would be valuable for research and development of big data systems.

The contribution of this paper is reference architecture for big data systems, and classification of related technologies and products/services. First, big data research, reference architectures, and use cases are surveyed from literature. Subsequently, the design of reference architecture for big data systems is presented, which has been constructed inductively based on analysis of the presented use cases. Finally, a classification is provided for the purpose of creating an overall picture of big data research, related technologies, products, and services.

The structure of the paper is as follows: Material and methods of the study are described in Section 2. Theoretical background is provided in Section 3. Design and construction of the reference architecture is presented in Section 4. Classification of big data technologies and commercial products/services, and survey of related work are provided in Section 5. The results are analysed in Section 6 and discussed in Section 7. A conclusion is provided in Section 8. The appendices include: a detailed description of the reference architecture (Appendix A), a detailed description of the

* Corresponding author.
E-mail addresses: pekka.paakkonen@vtt.fi (P. Pääkkönen), daniel.pakkala@vtt.fi (D. Pakkala).
[1] Tel.: +358 207227070.

http://dx.doi.org/10.1016/j.bdr.2015.01.001

research method (Appendix B), and references to surveyed commercial products and services (Appendix C).

2. Material and methods

The overall goal of this work is to facilitate realisation of big data systems. When a big data system is realised, important considerations include architecture design of the system, and utilization of underlying technologies and products/services [11]. The goals of this work are: a.) design technology independent reference architecture for big data systems b.) classify related technologies and products/services with respect to the reference architecture.

Reference architecture would be useful in the following ways: It should facilitate creation of concrete architectures [12], and increase understanding as an overall picture by containing typical functionality and data flows in a big data system. Classification of technologies and products/services should facilitate decision making regarding realisation of system functionalities. Also, it would be important to understand architecture and performance characteristics of related technologies. The following research questions are posed:

> The first research question: *What elements comprise reference architecture for big data systems?*
> The second research question: *How to classify technologies and products/services of big data systems?*

The reference architecture for big data systems was designed with inductive reasoning based on the published use cases described in Section 3.3 (research question 1). Particularly, functionality, data flows, and data stores of implementation architectures in seven big data use cases were analysed. Subsequently, reference architecture was constructed based on the analysis. The method for reference architecture design is described in detail in the Appendix. A literature survey was used for answering to the second research question.

3. Theory

Section 3.1 presents earlier surveys of big data. Research on big data reference architectures is presented in Section 3.2. The latest reports of big data use cases are introduced in Section 3.3. Finally, a summary of related work in presented in Section 3.4.

3.1. Big data research

Begoli [13] conducted a short survey of state-of-the-art in architectures and platforms for large scale data analysis. The survey covered adoption of related technologies, platforms for knowledge discovery, and architectural taxonomies. Chen et al. presented a comprehensive survey of big data [14]. The topics of the survey covers related technologies, generation and acquisition of data, storage, applications, and outlook to the future. Chen and Zhang also surveyed big data [11]. Their work focused on big data opportunities and challenges, techniques and technologies, design principles, and future research. Wu et al. provided a framework for big data mining [15]. The authors proposed HACE (Heterogeneous, Autonomous sources, Complex and Evolving relationships among data) theorem for characterizing big data. The authors also presented a three layer framework for big data processing, which is comprised of big data mining platform, semantics and application knowledge, and mining algorithms. Finally, Cuzzocrea et al. discussed Online Analytical Processing (OLAP) over big data, big data posting and privacy as part of big data research agenda [16].

3.2. Reference architecture for big data systems

A framework for design and analysis of software reference architectures has been presented [12]. The framework contains a multi-dimensional classification space, and five types of reference architectures. It is claimed that architecture design based on the classified reference architectures should lead to better success. Also, empirically-grounded design of software reference architectures has been presented [17]. The design approach is based on expected empirical material gathered with interviews, questionnaires, and document analysis. The procedure is a step-wise process, which consists of deciding a type for the reference architecture, selection of design strategy, empirical acquisition of data, construction of reference architecture, enabling of variability, and evaluation (see Appendix B for details).

Service-oriented reference architecture has been defined for enterprise domain [18]. However, in the big data context, there exist only few architecture proposals. Schmidt and Möhring [8] presented a service and deployment model for implementing big data pipeline to the cloud domain. Demchenko et al. presented a Big Data Architecture Framework, which consists of high-level description of big data lifecycle and infrastructure [9]. Doshi et al. presented reference architectures for integration of SQL and NewSQL databases in order to support different growth patterns in enterprise data traffic [19]. Zhong et al. proposed and validated big data architecture with high-speed updates and queries [20]. The architecture consists of in-memory storage system and distributed execution of analysis tasks. Cuesta proposed tiered architecture (SOLID) for separating big data management from data generation and semantic consumption [10]. Generalized software architecture was proposed for predictive analytics of historical and real-time temporally structured data [89]. Meier conducted design of reference architecture covering functionality in realised big data use cases (Master's Thesis [7]). The author initially defined requirements for reference architecture, conducted architecture design, and validated the presented architecture against published implementation architectures of Facebook, LinkedIn, and Oracle. The design was conducted in the empirically-grounded design framework for reference architectures [17,12].

3.3. Big data use cases

Many big data use cases have been published. Facebook, Twitter, and LinkedIn are examples in the social network application domain. Facebook collects structured and stream-based data from users, which is applied for batch-based data analysis [6]. Data scientists at Facebook can specify ad hoc analysis tasks in production or development environments for getting deep insight to the data. LinkedIn [1] also collects structured and stream-based data, which is analysed in development and production environments. Additionally, LinkedIn provides new services (e.g. "People you may know") for end users based on data analysis [1]. Twitter [3,21,22] handles mainly tweets, which have real-time processing requirements. Twitter also provides new services for end users e.g. "Who to follow" [23].

Netflix is a commercial video-streaming service for end users. Netflix collects user events, which are processed and analysed in online, offline, and nearline environments [2]. Video recommendations are provided for end users based on real time data analysis.

Also, network traffic has been analysed for getting value from data. BlockMon [4,24] is a high performance streaming analytics platform, which has been used for telemarketer call detection based on Call Data Records (CDR). Another application is monitoring of network traffic for execution of ad hoc Hadoop/MapReduce tasks [25,26]. The primary applications are web traffic analysis and

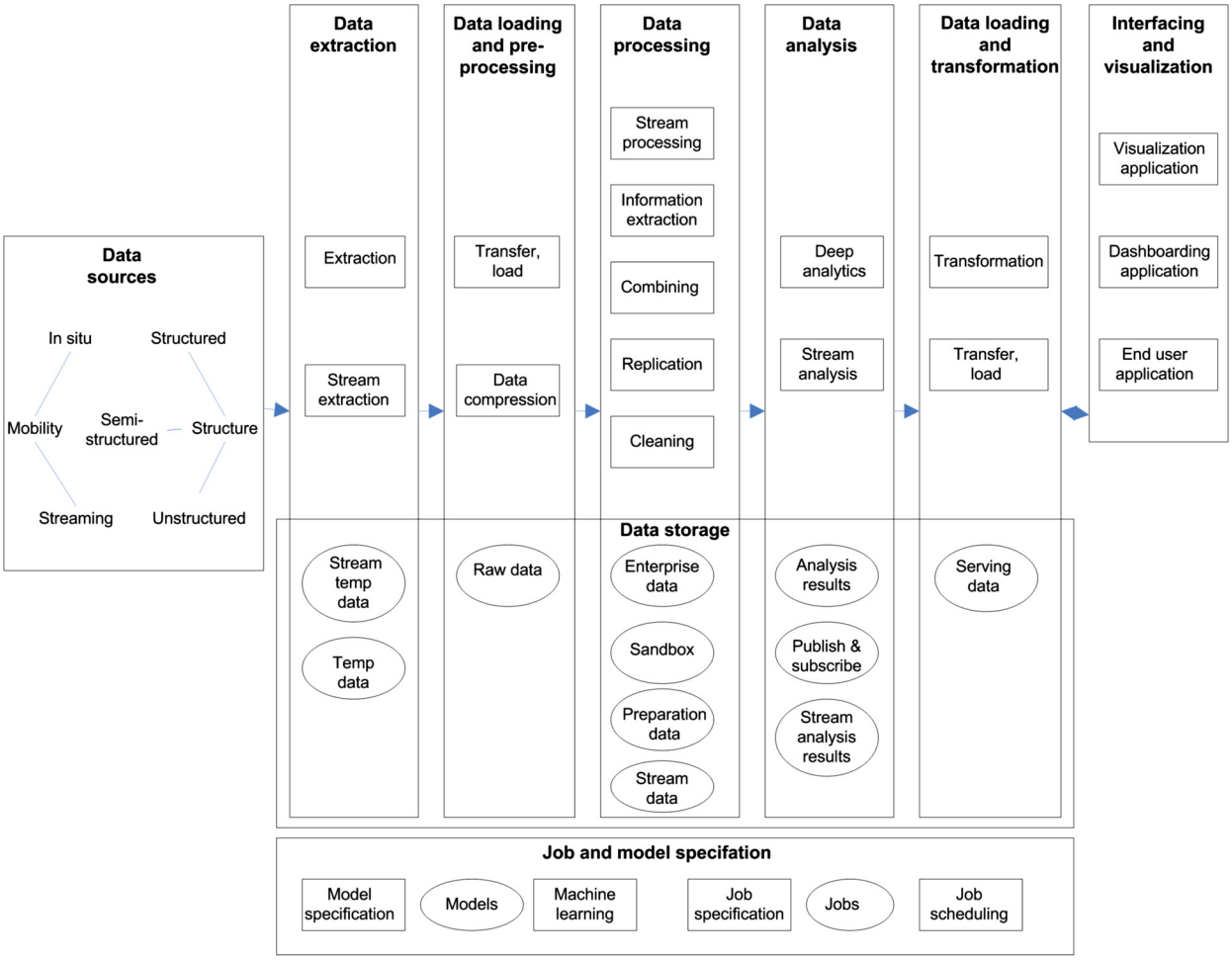

Fig. 1. High-level design of the reference architecture. Data stores are presented as an ellipsis, functionality as a rectangle, and data flows with arrows.

detection of distributed denial of service attacks. Finally, manufacturing process of digital displays has been improved based on data analysis [5].

3.4. Summary

Related work on big data architectures includes surveys of big data technologies and services (Section 3.1), initial design of reference architectures (Section 3.2), and reports on big data implementation architectures (Section 3.3). However, few reference architectures [7] for big data systems have been proposed (research question 1). Previous work includes either high-level models [8,9] or focus on subset of system functionality [10,19,20,89]. Also, big data surveys have been conducted, but a classification of related technologies, products and services is missing (research question 2).

4. Results

Section 4.1 presents design of the reference architecture. Particularly, different elements of the architecture are described. Section 4.2 presents how the reference architecture was constructed based on the published big data use cases.

4.1. Design of the reference architecture

Fig. 1 presents design of the reference architecture. In the figure functionalities (rectangle), data stores (ellipsis), and data flows (arrow) are applied for representation of the architecture. Data processing functionality has been presented as a pipeline, where data flows mostly from left to right. Similar functionalities have been grouped into functional areas. Data stores are presented together with the respective functional areas. Specification of jobs and models has been illustrated separately from the data pipeline. A detailed mapping between the reference architecture and use case has been provided in Appendix A (Fig. A.1). Concepts of the reference architecture are depicted in *italics* in the following description:

Data sources are defined in two dimensions, *mobility* and *structure* of data. First, *in situ* refers to data, which does not move. An example of *in situ* data is a Hadoop file to be processed with MapReduce. *Streaming* data refers to a data flow to be processed in real time, e.g. a Twitter stream. Second, structure of the data source is defined. *Structured* data has a strict data model. An example is contents of a relational database, which is structured based on a database schema. *Unstructured* data is raw, and is not associated with a data model. Web page content [92] or images can be considered as unstructured data. *Semi-structured* data is not raw data or strictly typed [91]. Other aspects of semi-structured data include irregularity, implicitness, and partiality of structure, and an evolving and flexible schema/data model [91]. Examples of semi-structured data include XML and JSON documents.

Extraction refers to input of *in situ* data into the system. When *in situ data* is *extracted*, it may be stored temporarily into a data store (*Temp data store*) or transferred, and loaded into a *Raw data store*. Streaming data may also be extracted, and stored temporarily (into *Stream temp data* store). Efficiency may be improved by

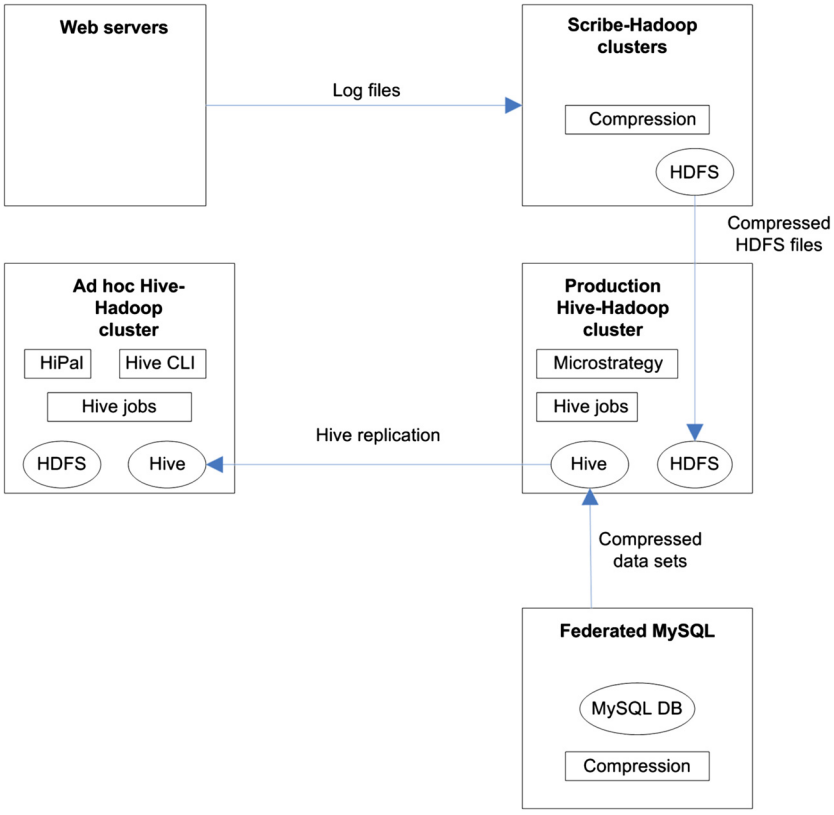

Fig. 2. Data analytics infrastructure at Facebook (adapted from [6]).

compressing extracted data before *transfer* and *load* operations. The purpose of the *Raw data store* is to hold unprocessed data. Data from the *Raw data store* may be *cleaned* or *combined*, and saved into a new *Preparation data store*, which temporarily holds processed data. *Cleaning* and *combining* refer to quality improvement of the raw unprocessed data. Raw and prepared data may be *replicated* between data stores. Also, new *information* may be *extracted* from the *Raw data store* for *Deep analytics*. *Information extraction* refers to storing of raw data in a structured format. The *Enterprise data store* is used for holding of cleaned and processed data. The *Sandbox store* is used for containing data for experimental purposes of data analysis.

Deep analytics refers to execution of batch-processing jobs for *in situ* data. Results of the analysis may be stored back into the original data stores, into a separate *Analysis results store* or into a *Publish & subscribe* store. *Publish & subscribe* store enables storage and retrieval of analysis results indirectly between subscribers and publishers in the system. Stream processing refers to processing of extracted streaming data, which may be saved temporarily before analysis. *Stream analysis* refers to analysis of *streaming* data (to be saved into *Stream analysis results*). Results of the data analysis may also be *transformed* into a *Serving data store*, which serve *interfacing and visualization applications*. A typical application for *transformation* and *Serving data store* is servicing of Online Analytical Processing (OLAP) queries.

Analysed data may be visualized in several ways. *Dashboarding application* refers to a simple UI, where typically key information (e.g. Key Performance Index (KPI)) is visualized without user control. *Visualization application* provides detailed visualization and control functions, and is typically realised with a Business Intelligence tool in the enterprise domain. *End user application* has a limited set of control functions, and could be realised as a mobile application for end users.

Batch-processing *jobs* may be *specified* in the user interface. The jobs may be saved and scheduled with *job scheduling* tools. Models/algorithms may also be specified in the user interface (*Model specification*). M*achine learning* tools may be utilized for training of the models based on new extracted data.

4.2. Construction of the reference architecture

In the following, construction of the reference architecture will be explained in detail. First, each big data use case is presented from architecture implementation point of view. Subsequently, mapping of each use case into the reference architecture is described. For most accurate information on implementation architectures, the reader is referred to the original cited work of the use cases, because the figures have been adapted for uniform presentation in this document.

4.2.1. Facebook

Data analytics infrastructure at Facebook [6] has been presented in Fig. 2. Facebook collects data from two sources. Federated MySQL tier contains user data, and web servers generate event based log data. Data from the web servers is collected to Scribe servers, which are executed in Hadoop clusters. The Scribe servers aggregate log data, which is written to Hadoop Distributed File System (HDFS). The HDFS data is compressed periodically, and transferred to Production Hive-Hadoop clusters for further processing. The Data from the Federated MySQL is dumped, compressed and transferred into the Production Hive-Hadoop cluster. Facebook uses two different clusters for data analysis. Jobs with strict deadlines are executed in the Production Hive-Hadoop cluster. Lower priority jobs and ad hoc analysis jobs are executed in Ad hoc Hive-Hadoop cluster. Data is replicated from the Production cluster to the Ad hoc cluster. The results of data analysis are saved back to Hive-Hadoop cluster or to the MySQL tier for Facebook users. Ad

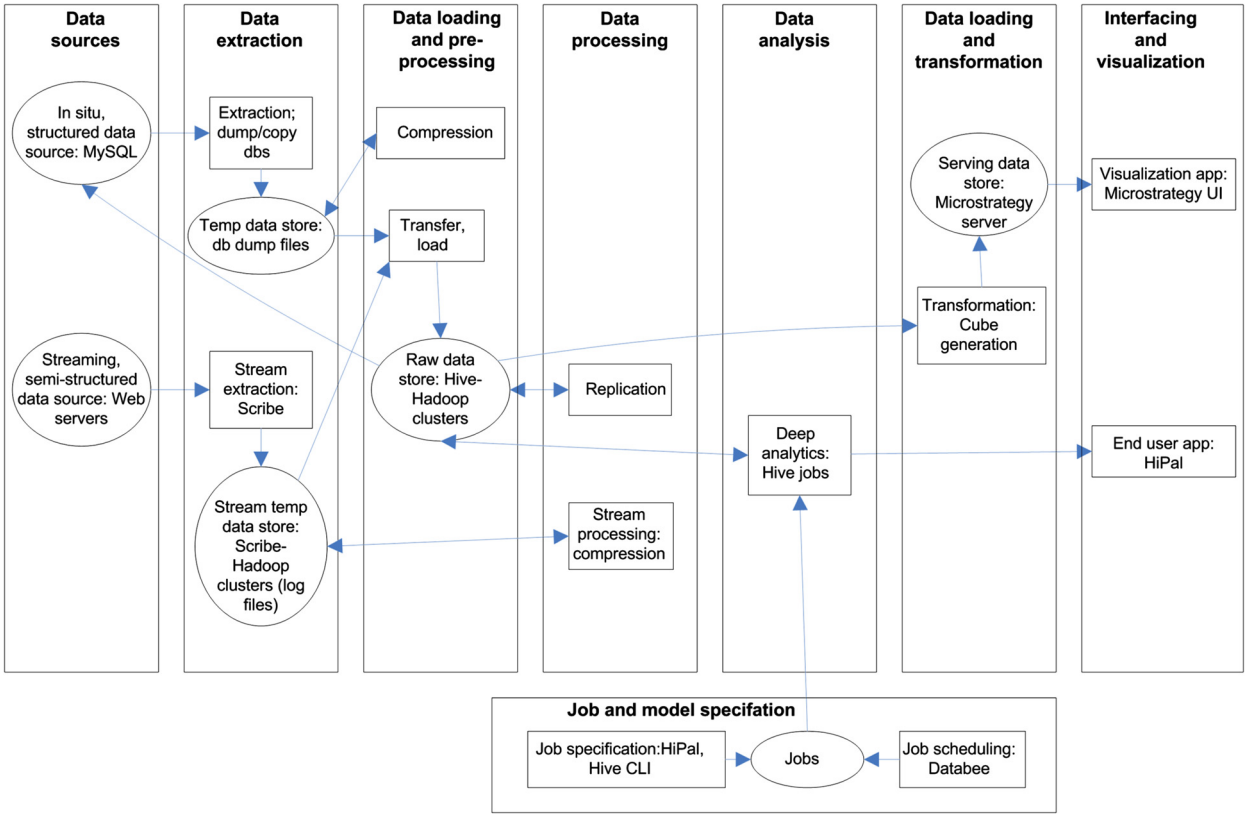

Fig. 3. Mapping between Facebook use case and the reference architecture.

hoc analysis queries are specified with a graphical user interface (HiPal) or with a Hive command-line interface (Hive CLI). Facebook uses a Python framework for execution (Databee) and scheduling of periodic batch jobs in the Production cluster. Facebook also uses Microstrategy Business Intelligence (BI) tools for dimensional analysis.

Mapping of the use case to the reference architecture has been described in Fig. 3. *Facebook extracts semi-structured, streaming* log data with Scribe (*stream extraction*). Scribe-Hadoop cluster acts as a *Stream temp data store*. Compression of stream-based data is referred to as *stream processing*. Facebook also extracts *in situ, structured* data, which is saved temporarily into Federated MySQL (*Temp data store*). Production Hive-Hadoop clusters act as a *Raw data store*, where both structured and streaming data is moved and saved (*transfer and load*).

Transfer and copying of data to the Ad hoc Hive-Hadoop cluster is modelled as *replication* of data. Execution of Hive jobs is modelled as Deep analytics i.e. batch-processing of data. *Job specification* refers to usage of HiPal or CLI user interface. HiPal is also used for visualization of results (*End user app*). *Job scheduling* corresponds to usage of the DataBee framework. Facebook also *transforms* (cube generation) data for further analysis with Microstrategy BI server (*Serving data store*) and *Visualization application*.

4.2.2. LinkedIn

The data analytics infrastructure at LinkedIn [1] has been presented in Fig. 4. Data is collected from two sources: database snapshots and activity data from users of LinkedIn. The activity data comprises streaming events, which is collected based on usage of LinkedIn's services. Kafka [27] is a distributed messaging system, which is used for collection of the streaming events. Kafka producers report events to topics at a Kafka broker, and Kafka consumers read data at their own pace. Kafka's event data is transferred to Hadoop ETL cluster for further processing (combining, de-duplication). Data from the Hadoop ETL cluster is copied into production and development clusters. Azkaban is used as a workload scheduler, which supports a diverse set of jobs. An instance of Azkaban is executed in each of the Hadoop environments. Scheduled Azkaban workloads are realised as MapReduce, Pig, shell script, or Hive jobs. Typically workloads are experimented in the development cluster, and are transferred to the production cluster after successful review and testing. Results of the analysis in the production environment are transferred into an offline debugging database or to an online database. Results may also be fed back to the Kafka cluster. Avatara [28] is used for preparation of OLAP data. Analysed data is read from the Voldemort database, pre-processed, and aggregated/cubificated for OLAP, and saved to another Voldemort read-only database [28].

LinkedIn extracts *semi-structured, streaming* event data, and *structured, in situ* database snapshots (Fig. 5). Kafka producer corresponds to *stream extraction* functionality. Kafka broker can be considered as a *Stream temp data store*. Copying of database snaphots is referred to as *extraction* of *in situ* data. Hadoop HDFS is a *Raw data store*, which holds both types of extracted data. *Combining* and de-duplication (*cleaning*) is performed to data, which is stored in Hadoop HDFS, before it is replicated. Copying of data to different clusters is modelled as *replication*. Processed data in the production environment is considered to be stored into an *Enterprise data store*. Data in the development environment is saved into a *Sandbox store*. The results of batch-based analysis (*Deep analytics*) in the production environment are saved either into the production Voldemort database (*Analysis results store*) or into the debugging Voldemort data store. Some of the results in the production environment are loaded back to the primary Kafka instance (to *Stream extraction*). Analysis results in the production environment are *transformed* with the Avatara-tool and loaded into another Voldemort data store (*Serving data store*) for serving OLAP queries from *End user*

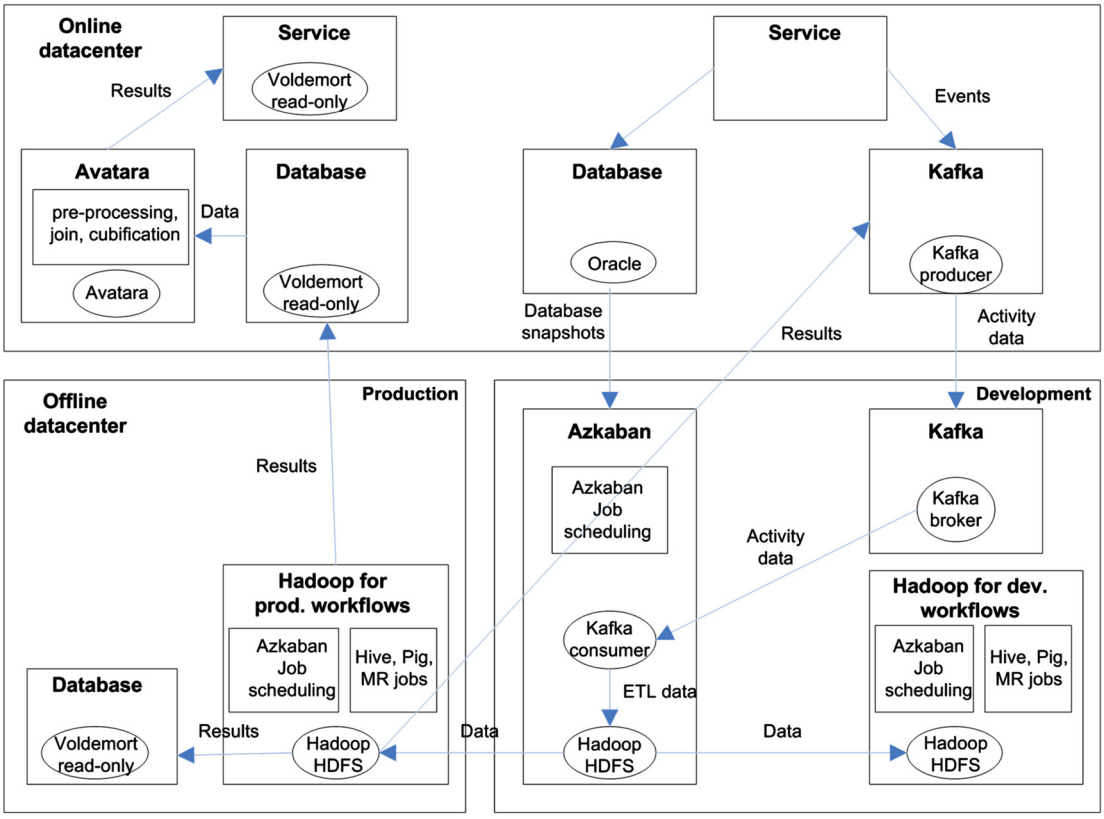

Fig. 4. Data analytics infrastructure at LinkedIn (adapted based on [1,28,27]).

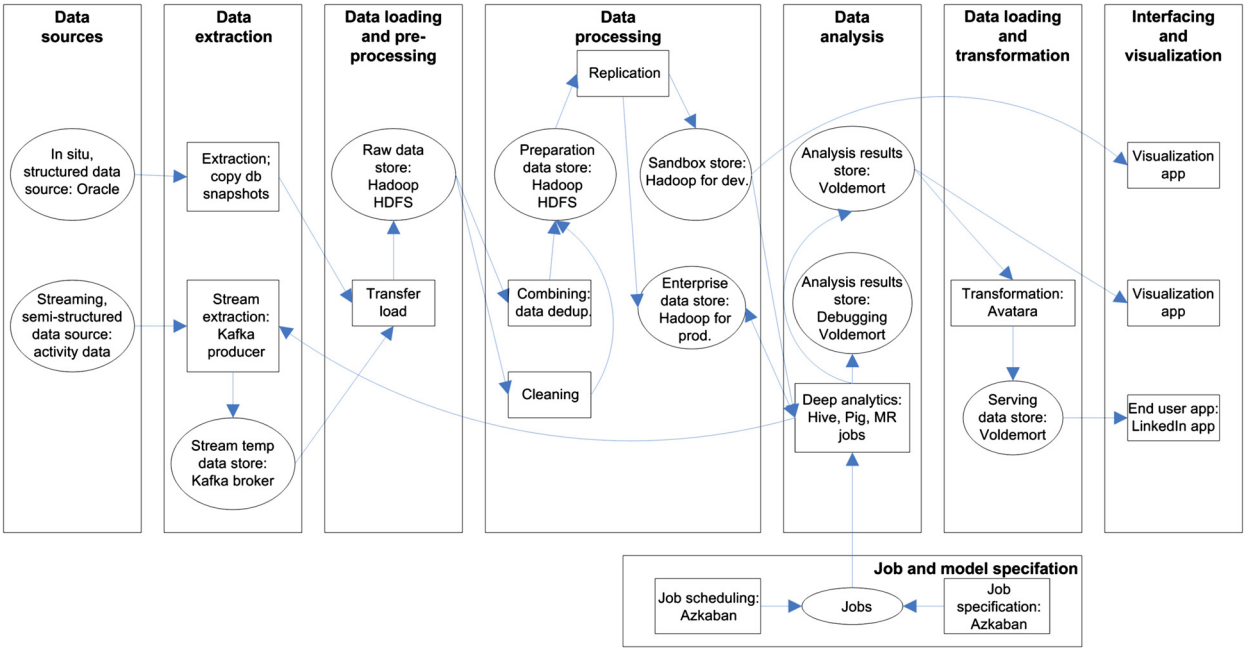

Fig. 5. Mapping between LinkedIn and the reference architecture.

applications (LinkedIn app). LinkedIn uses Azkaban for *specification* and *scheduling* of batch processing *jobs* in different domains.

4.2.3. Twitter

Twitter has published two different implementation architectures of their infrastructure. The first architecture was based on Hadoop batch processing, which led to performance problems in the processing of streaming data [3,21,22]. The second published infrastructure meets real-time performance requirements by replacing Hadoop-based processing with a custom-made solution [3], which is described in Fig. 6.

In the Twitter's infrastructure for real-time services, a Blender brokers all requests coming to Twitter. Requests include searching for tweets or user accounts via a QueryHose service. Tweets are input via a FireHose service to an ingestion pipeline for tokenization and annotation [29]. Subsequently, the processed tweets enter

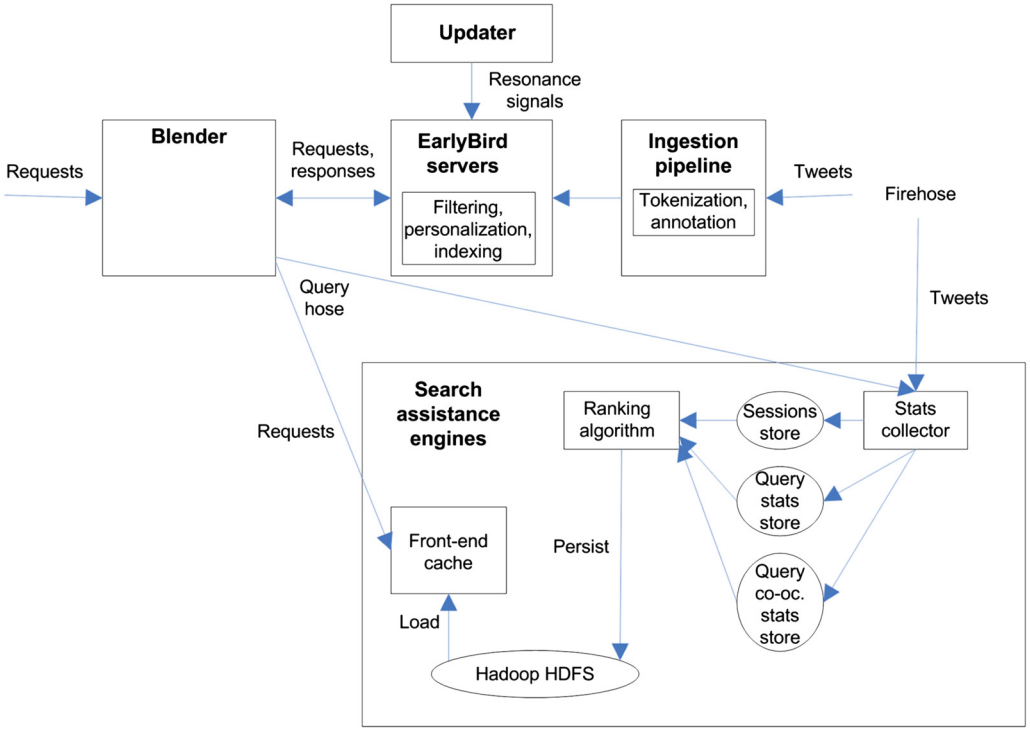

Fig. 6. Data analytics infrastructure at Twitter (adapted based on [3,29]).

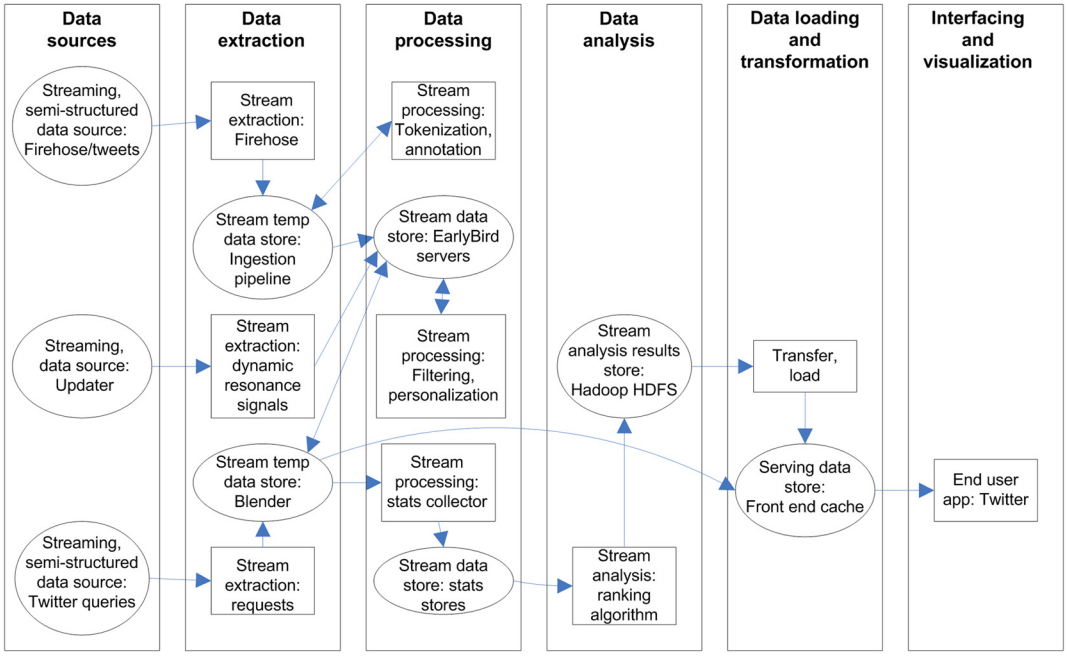

Fig. 7. Mapping between Twitter and the reference architecture.

to EarlyBird servers for filtering, personalization, and inverted indexing [29]. The EarlyBird servers also serve incoming requests from the QueryHose/Blender. The EarlyBird is a real-time retrieval engine, which was designed for providing low latency and high throughput for search queries.

Additionally, search assistance engines are deployed. Stats collector in the Search assistance engine saves statistics into three in-memory stores, when a query or tweet is served. User sessions are saved into Sessions store, statistics about individual queries are saved into Query statistics store, and statistics about pairs of co-occurring queries are saved into Query co-occurrence store. A ranking algorithm fetches data from the in-memory stores, and analyses the data. The results of analysis are persisted into Hadoop HDFS. Finally, Front-end cache polls results of analysis from the HDFS, and serves users of Twitter.

Twitter has three *streaming data sources* (Tweets, Updater, queries), from which data is *extracted* (Fig. 7). Tweets and queries are transmitted over REST API in JSON format. Thus, they can be considered as *streaming*, *semi-structured* data. The format of data from Updater is not known (*streaming data source*). Inges-

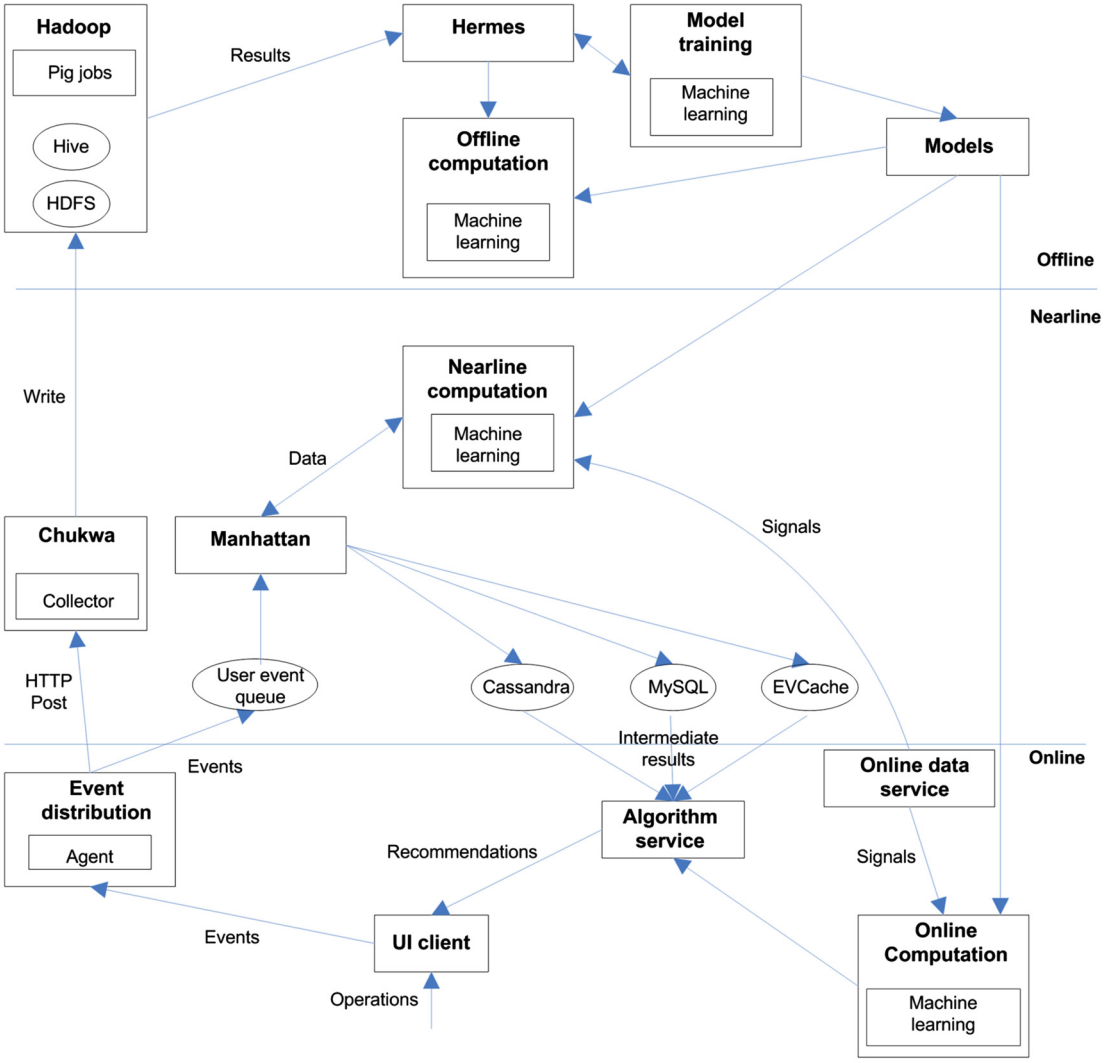

Fig. 8. Data analytics infrastructure at Netflix (adapted based on [2,30]).

tion pipeline and Blender can be considered as Stream *temp data stores*. Tokenization, annotation, filtering, and personalization are modelled as *stream processing*. EarlyBird servers contain processed stream-based data (*Stream data store*). Stats collector is modelled as *stream processing*. The statistical stores may be considered as *Stream data stores*, which store structured information of processed data. The ranking algorithm performs *Stream analysis* functionality. Hadoop HDFS storing the analysis results is modelled as a *Stream analysis data store*. Front-end cache (*Serving data store*) serves the *End user application* (Twitter app).

4.2.4. Netflix

Data analytics infrastructure at Netflix [2] has been presented in Fig. 8. The infrastructure is executed entirely in Amazon's cloud domain. Netflix has divided computation to online, nearline, and offline parts based on different real-time requirements. Services in the online computation have requirements for maximum latency, when responding to client applications. The nearline computation is similar to the online computation with the exception that computed results can be stored instead of immediate serving of end users. The offline computation has most relaxed requirements for timing.

End user interacts with Netflix by executing operations (e.g. Play, Rate etc.) in the user interface of the service. Recommendations are provided to the user based on other users' behaviour. Recommendation algorithms require data, models, and signals as input. The data is previously processed information, which has been stored into a database. The signals are fresh information from online services. The models are comprised of parameters, which are usually trained initially offline, but are enhanced based on incremental machine learning. Events from end users are distributed via the Chukwa framework [30,31] for the offline processing, or via user event queue for the nearline processing. Chukwa consists of agents, which transmit events in HTTP POSTs to collectors, which write data to HDFS file system. Manhattan is a distributed messaging system developed by Netflix. Hermes is a publish/subscribe framework, which is used for delivering of data to multiple subscribers in near real-time.

Netflix has two *streaming data sources*: online data service and Netflix user events (Fig. 9). The event-based data from Netflix users and signals from online data service are modelled as *streaming data*. The format of extracted streaming data is unknown. Chukwa agent can be considered as a *stream extraction* process, and Chukwa collector as a *Stream temp data store*. Hadoop HDFS can be modelled as a *Raw data store*. Execution of offline Pig *jobs* is modelled as *Deep analytics*. Hermes is a *Publish & subscribe store* for storing and retrieving of offline analysis results.

User data queue is modelled as a *Stream temp data store*. Data transformation by Manhattan is modelled as *Stream processing* for near-line computation. Intermediate analysis results are

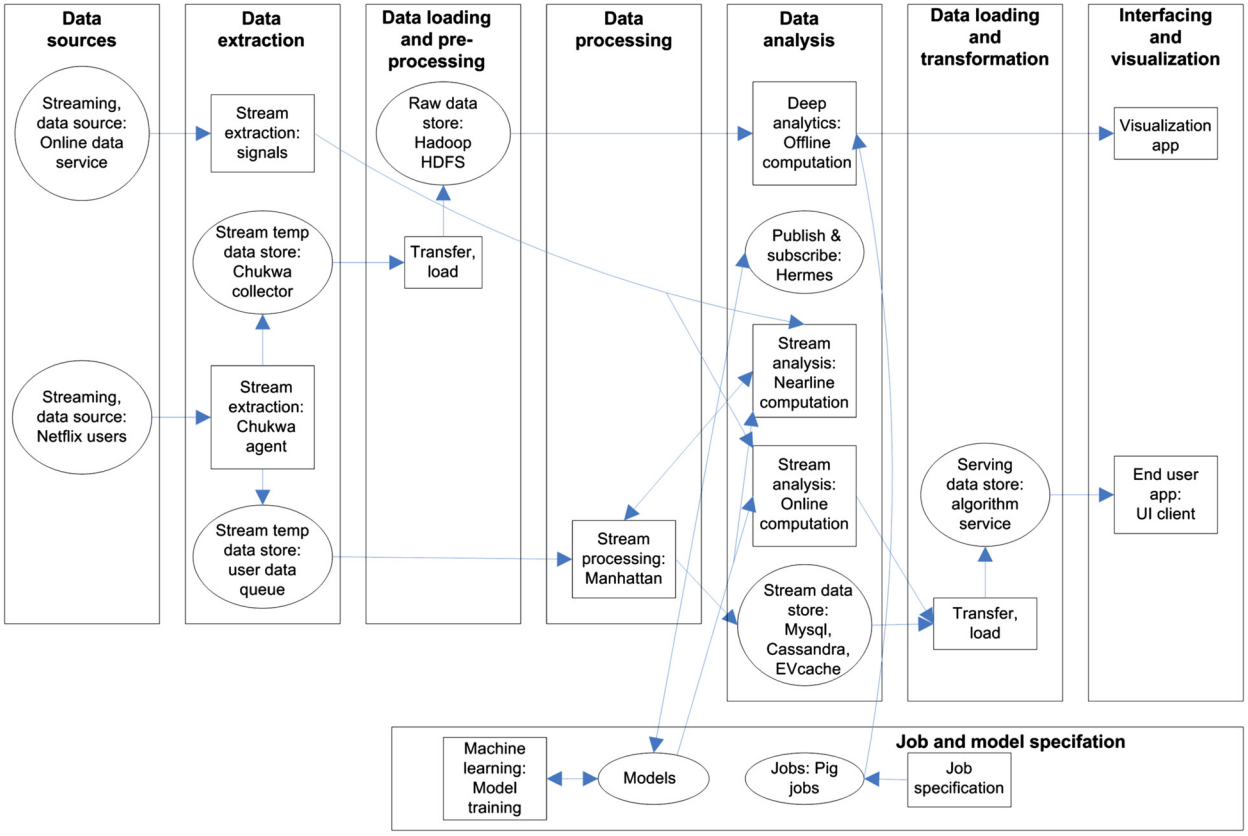

Fig. 9. Mapping between Netflix and the reference architecture.

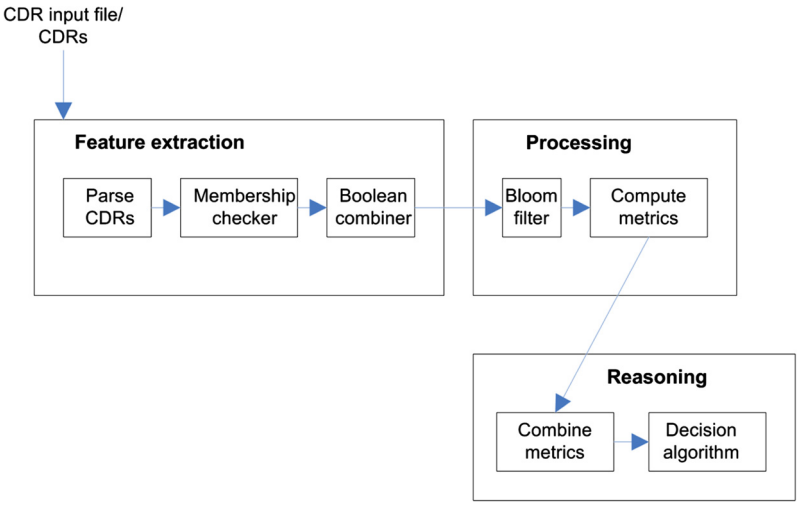

Fig. 10. BlockMon streaming analytics platform with VoIPStream (adapted based on [4,24]).

stored (processed streaming data) into *Stream data stores* (Cassandra, MySQL, EVCache). Online and nearline computation is modelled as a *Stream analysis* process. Algorithm service acts as a *Serving data store* for Netflix UI clients (*End user application*). Additionally, *Models* are trained with *machine learning* algorithms.

4.2.5. BlockMon

BlockMon [4,24] is a high performance streaming analytics platform (Fig. 10). BlockMon consists of small interconnected blocks, each of which performs certain type of processing. The interconnected whole can be programmed to perform a larger job. In this paper, VoIPStream as an application of BlockMon is presented for detection of telemarketer calls in real-time.

VoIPStream has been divided into feature-extraction, processing, and reasoning parts. First, VoIPStream takes a stream of Call Data Records (CDR). The CDRs are parsed and features are gathered for basis of the analysis. A processing part keeps track of the overall behaviour of each user by computing different metrics with bloom filters. Finally, the reasoning part combines calculated metrics. A threshold-based decision algorithm determines, if the user is acting like a telemarketer.

BlockMon has one *streaming, structured data source* (*CDRs*) (Fig. 11). Parsing of CDRs, checking of memberships, and Boolean

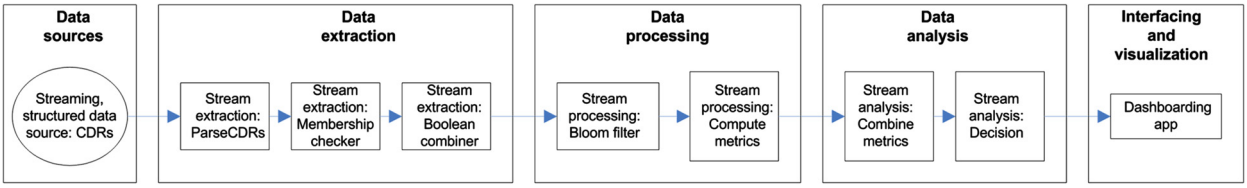

Fig. 11. Mapping between BlockMon and the reference architecture.

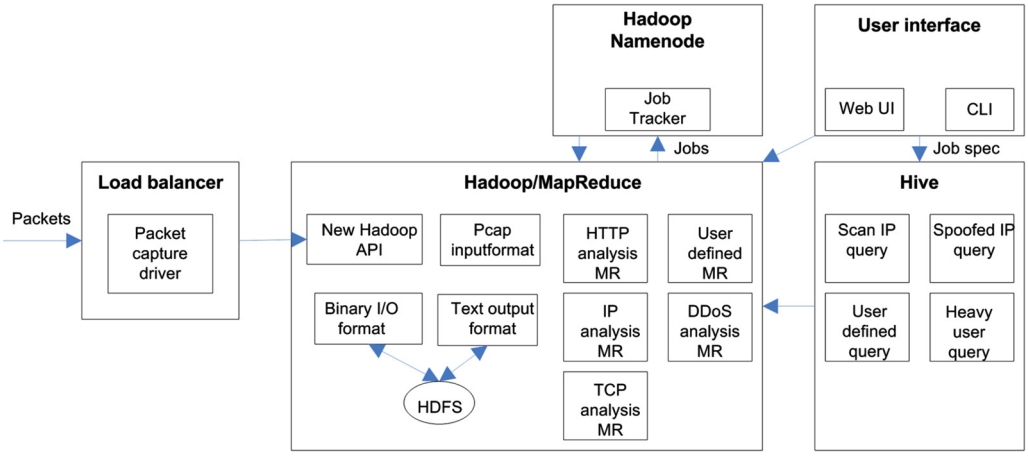

Fig. 12. Network traffic measurement with Hadoop (adapted based on [25,26]).

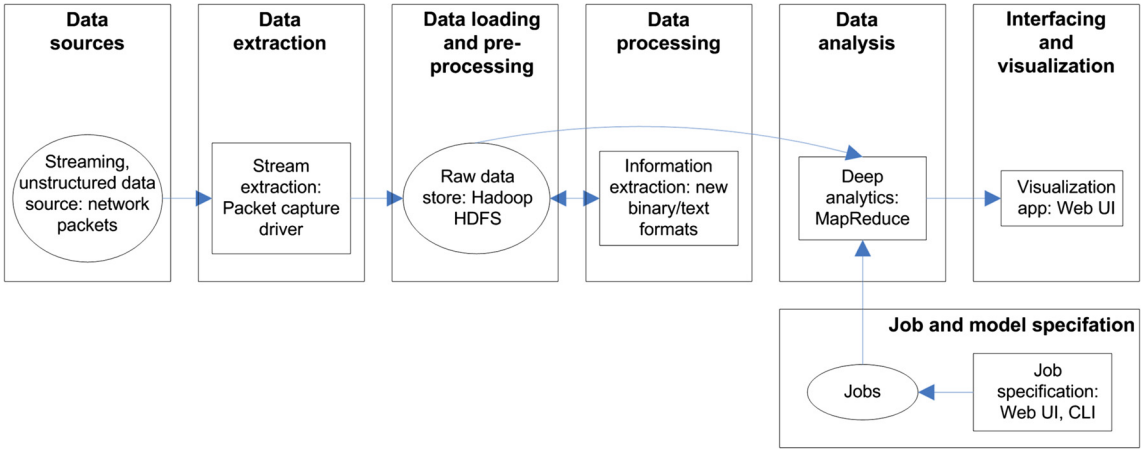

Fig. 13. Mapping between Network measurement use case and the reference architecture.

combining can be considered as *stream extraction* functionalities. Bloom filtering, and computation of metrics are considered *stream processing*. Combining of metrics and decision making are modelled as *stream analysis*. The result of analysis is displayed in a simple *Dashboarding application* (visualization with traffic lights in [24]).

4.2.6. Network measurement

Network traffic measurement and analysis solution [25] has been presented in Fig. 12. The solution consists of a traffic collector and Hadoop based traffic analysis tools. The traffic collector consists of a load balancer and HDFS Datanodes. The load balancer has a packet capturer for monitoring of traffic from a router or another network device. It forwards the captured packets to HDFS Datanodes based on a flow level hashing function. The received packets are saved into HDFS with a new Hadoop API to support libpcap format. Also, new formats have been implemented on top of HDFS to support processing of HDFS data in the native libpcap format [26].

End user can specify jobs/queries interactively in the user interface. The specified jobs are mapped either into Hive or MapReduce jobs on a lower level. MapReduce/Hive based tools have been developed for execution of different type of network analysis.

Packet capture driver is considered as an extractor of *streaming, unstructured* data (Fig. 13). Hadoop HDFS is modelled as a *Raw data store*. New binary/text formats are considered as *extraction of information* from the raw data i.e. structure is created around unstructured raw data. MapReduce tasks are modelled as *Deep analytics* functionality. The web interface is applied for specification of the batch processing jobs (*Job specification*) and visualization of analysis results (*Visualization app*).

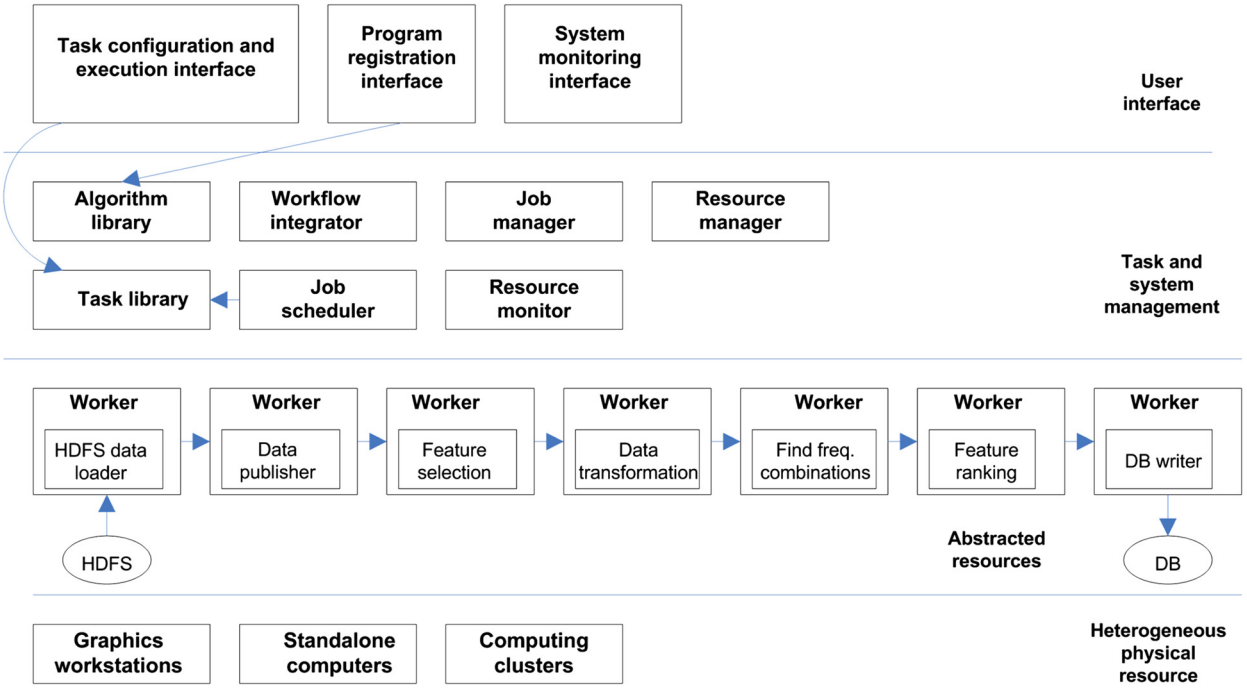

Fig. 14. FIU-Miner data mining platform in a distributed environment (adapter based on [5]).

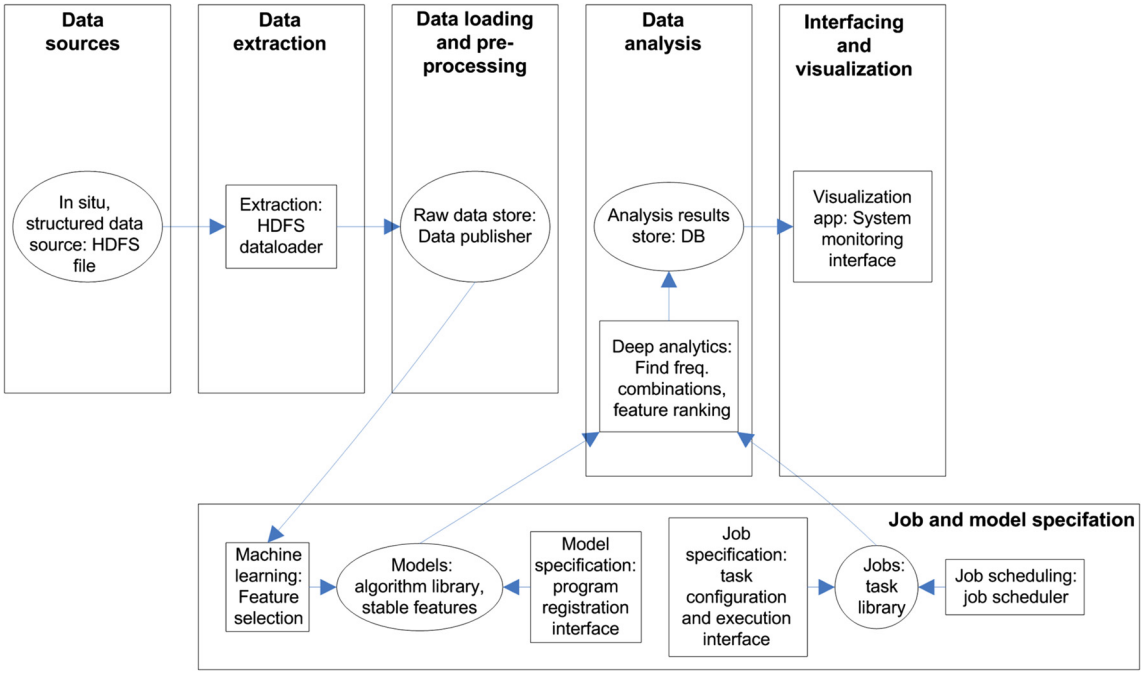

Fig. 15. Mapping between FIU-Miner and the reference architecture.

4.2.7. FIU-Miner

FIU-Miner [5] is a system for facilitating complex data analysis processes. Adapted architectural view of FIU-Miner has been presented in Fig. 14. The system consists of four architectural layers. User interface is used for task configuration and execution, importing of data mining programs to the algorithm library, and monitoring of resource utilization. Task and system management layer consists of algorithms, integration of workflows, and scheduling of jobs. It also includes tracking of job execution, and monitoring and management of workers.

The execution of jobs is performed in the abstracted resources layer, which has been demonstrated with a case study for optimizing manufacturing process of digital displays [5]. In the demonstration a dataset of displays is loaded from HDFS, and is dispatched into three feature selection algorithms [32]. The selected features are transformed, combined, ranked and stored into a database. The extracted feature combinations are highly related to yield ratio of the manufacturing process, which depends on parameter setting of production equipment.

HDFS file can be considered as an in situ, *structured data source* (Fig. 15). HDFS dataloader extracts information from HDFS file, which is stored into a *Raw data store* (Data publisher). Feature selection functionalities can be considered as *machine learning*, as the models are mined with different feature selection algorithms.

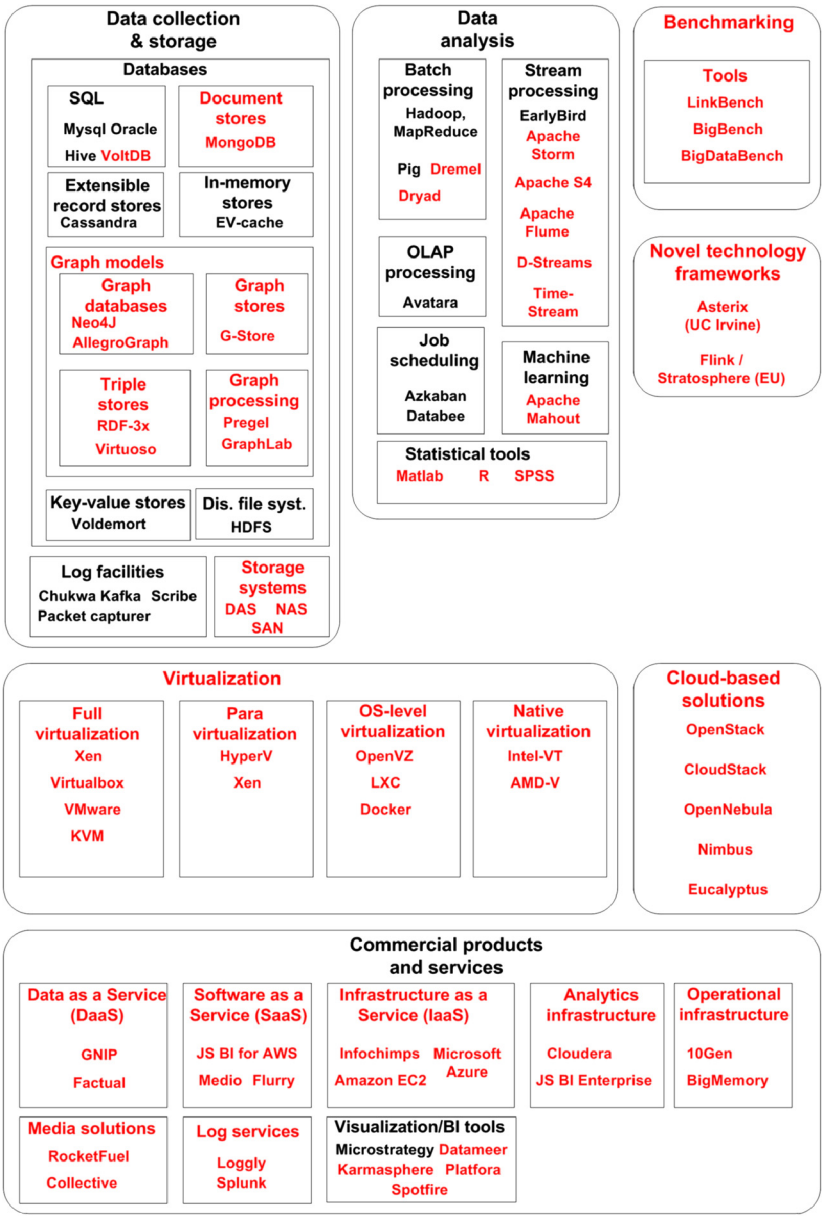

Fig. 16. Classification of big data technologies, products, and services. Technologies/products/services of the reported big data use cases (black) are depicted separately from the related work (red). (For interpretation of the references to colour in this figure, the reader is referred to the web version of this article.)

The mined stable features (*Models*) are saved. Finding of frequency combinations, and ranking of features can be modelled as *Deep analytics* functionality. Analysis results are stored into the database (*Analysis results store*). Task configuration and execution interface can be considered as *Job specification*. Program registration interface can be used for *model specification*. System monitoring interface is used for visualization (*visualization app*).

5. Review of big data technologies

5.1. Classification of technologies, products and services

An initial classification of technologies, products and services was created based on the reviewed big data use cases (Fig. 16), which are described in the following: Data in the use cases was typically collected into databases or log files. Relational databases were applied for storage of important user related data (Facebook's MySQL, LinkedIn's Oracle). NoSQL databases or in-memory stores were used for intermediate or final storage of data analysis results (LinkedIn's Voldemort, Netflix's Cassandra and EV-cache). Special log facilities were used for storage of stream-based data (Facebook's Scribe, LinkedIn's Kafka, Netflix's Chukwa, Packet capture driver in the Network measurement case). Structured data was typically saved into a distributed file system (e.g. Hadoop HDFS).

Data processing technologies of the use cases can be classified into batch and stream processing. Jobs with real-time constraints require special technologies and algorithms (Twitter's ranking algorithm and the EarlyBird architecture). Batch processing was used for jobs with less strict timing requirements (Facebook's, LinkedIn's, and Netflix's MapReduce, Hive and Pig scripts). Jobs for batch processing were scheduled with special tools (LinkedIn's Azkaban, Facebook's Databee). OLAP processing (LinkedIn's Avatara) transformed data for serving of OLAP queries. Processed data was also visualized with commercial Business Intelligence (BI) tools (Facebook uses MicroStrategy).

Table 1
Categorization of stream-processing techniques.

Publication	Technique	Programmer's view	Reported processing performance
MapReduce Online [34]	Pipelining between jobs: map task pushes data to reducers; Online aggregation of jobs.	MapReduce+optional "flush" API	–
Faster data analytics [33]	Distributed shared memory for storing of intermediate MR results; RDMA for reduction of network latency.	MapReduce	Initial simulation results
MapUpdate: Muppet [35]	map and update functions as streams; distributed slates for storage of intermediate results.	MapUpdate	Muppet @ Twitter and FourSquare: 100 M tweets/day, 1.5 M check-ins/day; Latency $<\sim$2 s.
S4 [36]	Distributed cluster of Processing Elements	Processing Elements API	$\sim$1600 clicks/s
SQLStream [38]	In-memory storage; streaming SQL	SQL API	15 000 CDRs/s
TimeStream [39]	Streaming DAG	StreamInsight API	Sentiment analysis of 9600 tweets/s; latency $\sim$2 s.
D-Streams/Spark [40]	Stream divided into short, stateless, deterministics tasks in Spark streaming engine.	D-Stream API in Spark Streaming with Scala/Java programming language	640 000 Grep records/s, 250 000 TopKCount operations/s; latency 1–2 s
Naiad [41]	Timely Dataflow: stateful vertices send/received messages along directed edges.	Timely Dataflow API; SQL, MapReduce, LINQ	32 000 Tweets/s; latency $<$1 s
Integrated Execution Platform [42]	Integration between System-S (IBM) and Hadoop.	SPADE (IBM)	Stream and batch processing of Tweets: latency, CPU usage reported.

5.2. Related work

The analysed use cases cover only partly research and commercial services in the big data area (Fig. 16). The survey of related work focuses on stream processing, graph stores, benchmarking, virtualization and cloud solutions, business intelligence and visualization tools, novel technology frameworks, and commercial services. These particular categories were focused on, because the cited big data use cases did not cover them, but they were important for answering to the second research question.

5.2.1. Stream processing

Stream processing solutions have been analysed in terms of processing technique, programmer's view, and performance characteristics (Table 1). First, many extensions to the MapReduce framework have been developed for supporting streaming applications (Table 1), due to its limitations [33]. MapReduce Online [34] supports online aggregation of single or multiple jobs in order to get intermediate results of jobs. Also, pipelining enables pushing of data from map task to reducer task for enabling streaming capability. Mavani and Ragha proposed usage of distributed shared memory for storing of intermediate MapReduce results, and Remote Direct Memory Access (RDMA) for reduction of network latency [33]. Example RDMA networks include InfiniBand or Quadrics [33]. MapUpdate is another model for processing of fast data [35]. Similar to MapReduce, the developer writes map and update functions, but processing is performed on streams, which may never end. MapUpdate uses distributed in-memory slates for storing intermediate results of update functions. Muppet, an implementation of the MapUpdate model, has been used for processing of Twitter and FourSquare streams [35]. Muppet has enabled quick development of stream processing applications, and achieved low latency and high scalability. Also, many other MapReduce-based alternatives exist, which have been reviewed by Mavani and Ragha [33].

S4 is a distributed platform for stream processing [36]. Computation in S4 is performed by Processing Elements (PE), each associated with a keyed attribute, and a value. PEs may be independent or may be connected by consuming results produced by other PEs. Processing Nodes (PN) host PEs, and are responsible for listening to events, and invoking of computation in PEs based on the events. A separate communication layer abstraction manages clusters of PNs, and maps logical elements to physical nodes. S4 provides an API for programming stream processing as an abstraction of PEs. Online computation of click-through-rates was performed with a peak performance of 1600 events/s [36]. Offline computation in 8-node cluster handled $\sim$12 000 events/s with less than 1% relative error [36]. Also, scalability, resource usage, and fault tolerance of S4 has been analysed [37].

SQLStream is a product, which is based on parallel in-memory storage of log data. It offers SQL standard-based API to the programmer. Processing is based on a pipeline of streaming SQL operations. 15 000 CDRs per second was processed for a SIP trunking provider [38]. Streaming data can also be saved into Hadoop Hive for deeper analysis.

Qian et al. proposed TimeStream, which is a distributed system for processing of low latency streaming data on commodity machines [39]. It uses streaming directed acyclic graph (DAG) computation at a lower level. The programming model of the system is based on Microsoft's StreamInsight technology for complex event processing applications. It has been extended for distributed execution supporting parallel execution, fault tolerance, and dynamic reconfiguration. The system is able to perform sentiment analysis of 10 000 Twitter tweets/s with a 2 s delay in a computer cluster.

Zaharia et al. have developed discretized streams (D-Streams) for enabling fault-tolerant streaming computation [40]. D-Streams structures stream computations as a set of short, stateless, and deterministic tasks, which allows powerful recovery techniques. Tasks are stored in-memory as resilient distributed datasets. D-Streams has been implemented based on Spark processing engine. It has high per-node throughput, scales up to 100 nodes, and achieves sub-second fault recovery and latency. D-Streams is developed further as part of the Apache Spark project.

Naiad is a new distributed system for executing parallel cyclic dataflow programs [41]. It is based on a low level computational model (timely dataflow), and modelling with directed graphs. In the solution stateful vertices transfer messages along directed edges [41]. Naiad is a distributed implementation of the timely

dataflow, and enables programming with several higher level abstractions (e.g. MapReduce, LINQ). The tests indicate that it can achieve equivalent performance with other similar stream processing systems.

Matsuura [42] proposed an integrated execution platform, which is able to handle both stream and batch processing jobs while satisfying Service Level Agreement (SLA) for stream processing applications. The system is based on distributed data stream processing middleware (System S by IBM research) and Apache Hadoop technologies. In the proposal a dynamic load balancing component allocates computing nodes for stream or batch processing based on a scheduling strategy. Other important stream processing technologies include Apache Storm and Apache Flume.

The review indicates that existing stream processing techniques are based on MapReduce extensions or modifications [34,33,35], streaming SQL operations [38], dividing of streams into small deterministic tasks [40], modelling based on acyclic graphs [39,41], distributed system of processing elements [36], and simultaneous batch and stream processing solutions [42]. From the programmer's viewpoint MapReduce [34,33] and its extensions [35], SQL API [38], or completely new APIs [40–42,36] can be utilized. Performance measures of the solutions mainly included latency and throughput, and Twitter was used as the main streaming reference use case.

5.2.2. Graph models

Graph models are based on the semantics of graph theory [43]. A graph data model can be characterized by graph representation, data manipulation expressed by graph transformation, and integrity constraints enforcing data consistency [43]. Graph modelling is applied in big data area, when information about data interconnectivity is more important than data itself [43]. This enables more natural modelling of data, and direct referral to the graph structure by complex queries [44]. These are important properties for applications using graph models such as social or biological networks [43].

Implementations of graph database models have been analysed and categorized by Angles [45]. Graph databases contain major components of a database system such as database language, query optimizer, and database engine. Graph stores provide only basic facilities for storing and querying of graphs. Related graph technologies do not store graphs, but use graph notion in the implementation. Related graph technologies include triple stores of the Resource Description Framework (RDF). Also, web and document oriented databases, which use graph algorithms or structures in implementation, may be considered as related graph technologies [45]. Furthermore, Angles compared graph databases and stores in terms of data structures, query languages, integrity constraints, and support for querying [45].

Typical application needs in graph processing include online query processing with low latency requirements, and offline graph analytics with high throughput requirements [46]. XGDBench was used for benchmarking online graph databases with synthetic graph models, which indicated that OrientDB had best performance, when compared to the alternatives (AllegroGraph, Fuseki, Neo4j, Titan) [47]. Angles proposed a micro-benchmark for testing social networking related databases [48]. Graph databases (DEX and Neo4j) achieved best performance, when compared to RDF stores or relational database (RDF-3x, Virtuoso and PostgreSQL). Abreu et al. [49] analysed graph databases (DEX, Neo4j, HypergraphDB) and RDF engine (RDF-3x) with Berlin SPARQL benchmark for consuming and mining of linked data. None of the tested technologies outperformed others, but RDF-3x exhibited best performance in most tasks.

RDF stores can be considered as a sub group of graph stores. Huang et al. [50] presented a scalable distributed RDF data management system, which is based on partitioning of graphs. It enables triples close to each other in the graph to be stored in the same RDF store. The approach significantly increases performance of serving slow SPARQL queries, when compared to alternatives (single node RDF-3x, SHARD). A recently developed graph engine manages RDF data in native graph form in a distributed in-memory store [51,52]. It achieves much higher performance for online SPARQL queries or offline computation than alternatives, which manage data in triple stores or as bitmap matrices [51,52].

Specific graph processing systems have also been developed. Google published Pregel, a distributed system for large-scale graph processing [53]. Pregel exposes a Vertex API for programming of graph processing jobs. Apache Giraph is an open source graph processing engine inspired by Pregel. Distributed GraphLab [54] is an alternative approach, which enables distributed, asynchronous, and dynamic processing of graphs. GraphChi [55] is a disk-based system for computing graphs with millions of edges. It divides big graphs into smaller parts by utilizing a parallel sliding windows method, which enables simultaneous execution of data/graph mining and machine learning tasks on a single computer.

Twitter's "Who to Follow"-service [23] is a real world example involving graph processing. Customers are served by storing user graph on a single in-memory database (Cassovary) [23]. The server handles OLAP queries such as user recommendations, whereas Online Transaction Processing (OLTP) based graph manipulations are handled by another graph database (FlockDB). Persistent data from FlockDB is loaded daily into Cassovary server due to the size of the graph.

5.2.3. Business intelligence and visualization

Traditional Business Intelligence relies on Extract-Transform-Load (ETL) tools for input of data into warehouse servers [56]. Typically data is processed and stored to mid-tier servers for providing a multi-dimensional view on OLAP-server, enterprise search engines, or data mining engines for enabling in-depth analysis of data. Finally, front-end applications provide user interface and functions such as searching, performance management, or tracking of key performance indicators.

However, requirement for faster decision making has created a need for further development of BI tools and technologies. Choo and Park have proposed computational methods for improving performance of visual analytics [57]. The solution is to exploit discrepancies in human perception and screen space by utilizing low-precision computation, iteration-level interactive visualization, and iterative refinement of computational results [57]. Another big data visualization challenge has been encountered with electronic microscope, which may involve long delays in the data pipeline [58]. As a solution Beyer et al. have presented a system for exploration of peta-scale data volumes in 3D. The system is based on visualization-driven volume data construction and on a multi-resolution virtual memory scheme.

In addition to academic work on visualization, several commercial tools are available. Vendors have been categorized to Table 2 in terms of UI capabilities, supported execution environments and data sources, capabilities for data analysis, and support for external statistical tools.

Most of the reviewed solutions offer visualization with a web browser. Also, mobile applications may be provided for extending the customer experience (Tableau, QlikView, SAS). Most of the products can be executed in enterprise or cloud domain. Some product offerings have been associated with the execution environment. For example DataMeer and Tableau provide separate products for desktop, server, and cloud environments. Supported

Table 2
Analysis of commercial visualization tools. See Appendix C for references.

Vendor	UI capabilities	Execution environment	Data sources	Data analysis capability	External statistical interfaces
Karmasphere	Web browser	Physical HW, cloud	Delimited (CSV, TSV, XML etc.), text, extended text, sequence, binary.	Analytical functions, batch analysis (Hadoop)	–
Datameer	Web browser (HTML5)	Desktop, server, Hadoop cluster	Social media, RSS, RDBMS, NoSQL, web/app logs, Java	Batch analysis (Hadoop)	R, SPSS, PMML
Platfora	Web browser (HTML5)	Physical or cloud. Requires access to HDFS or Amazon S3.	Delimited text, Hive tables	See "Analytics engine", batch analysis (Hadoop)	–
Tableau	Web browser. Mobile applications for Android, iPad, iPhone.	Desktop, server, cloud (online). Runs on Windows. Virtual env.	Databases, cubes, data warehouses, files and spreadsheets.	See "Tableau Desktop"	R
QlikView	Web browser, QlikView Mobile for iOS. "Business discovery" for organizations.	Physical. Runs on Windows.	Data warehouses, Excel files, SAP, Salesforce.com	See "Qlikview Developer"	–
SAS Visual Analytics	Mobile apps for iPad and Android	Physical, cloud, SaaS	Excel, delimited text, SAS data, Twitter stream, Oracle tables	See SAS Visual Analytics Designer	–
Streambase	Web browser, mobile client access via Restful API	Windows, Linux	Various streaming data sources (news, social media etc.)	See "LiveView Desktop"	Matlab, NVidia, R, kdb+QDBC

data sources are heterogeneous, although StreamBase differentiates by providing support for analysis of streaming data with low latency requirements. A few vendors enable integration by visualizing results of Hadoop-based analysis. Some vendors also provide adapters for integration with external statistical tools.

5.2.4. Big data benchmarking

Chen et al. analysed MapReduce traces of business critical deployments at Facebook and Cloudera [59]. They found out that most workloads are diverse, typically driven by interactive analysis, and apply query-type of programming frameworks on top of MapReduce. Thus, big data benchmarks should characterize diverse patterns of workloads.

Currently new benchmarking technologies for big data systems are being developed, which have been surveyed by Qin and Zhou [60]. The review indicated that many benchmarks exist for big data related technologies such as MapReduce, NoSQL databases, graph databases [47], Hadoop, and relational databases [60]. However, a common benchmark covering complex big data systems and workloads [59] is needed, which was also concluded by Wang et al. in a review of big data benchmarks [61].

Ghazal et al. have developed BigBench, an end-to-end benchmark for simulation of comprehensive workloads [62]. Data model of the benchmark adopts a structured part from an existing TPC-DS benchmark. Semi-structured and unstructured data components have been implemented with Parallel Data Generation Framework [63]. The proposal covers specification of velocity, variety, and volume, which are important characteristics of big data. Performance of the benchmark was tested against Teradata Aster DBMS in a clustered environment. An alternative approach is driven by Big Data Benchmarking Community [64]. It aims at defining an end-to-end application layer benchmark, which would be evolved with a concurrent benchmarking model. Armstrong et al. have developed LinkBench, which is a benchmark based on Facebook social graph [65]. It reflects real-world database workloads for social applications. The benchmark consists of a loading phase for populating a graph store with synthetic data, and execution phase for running the specified queries and collection of statistics. Wang et al. presented BigDataBench [61] for characterizing big data workloads. First, real-world data sets were gathered from social networking, e-commerce, and search engine use cases. Then, 19 workloads were chosen covering offline and real-time analytics, and online services. BigDataBench has been used for benchmarking of big data systems with different hardware platforms [66].

The review indicated that various benchmarks have been developed for testing of big data technologies [60]. Also, the first end-to-end benchmarks are being developed for characterizing big data workloads [62,65,64,61].

5.2.5. Virtualization and cloud-based solutions

Virtualization has been developed for abstracting HW and system resources to enable execution of multiple operating systems (OS) (Fig. 16). Full virtualization [67] aims at hardware emulation. An unmodified OS is used, in which a hypervisor controls execution of privileged operations. Paravirtualization requires modifications to the virtualized OS and coordination between the virtual OS and hypervisor regarding operations [67]. The benefit is increased performance over full virtualization, which is enabled by execution of demanding tasks in the host OS. OS-level virtualization approach does not require a hypervisor. Instead, the underlying OS is modified to enable execution of multiple instances of the OS on a single machine [67]. Linux versions of OS-based virtualization have been referred to as container-based virtualization solutions [68]. The main advantage is performance, which is close to execution of a native OS [67]. The disadvantage is reliance of multiple VMs on a single OS kernel, which may create problems in case of crashes. An interesting technology is Docker, which automates deployment of applications in standardized execution environments, by utilizing container-based virtualization. Native virtualization uses hardware support in a processor to enable execution of multiple unmodified OSs on the host processor.

Performance of different virtualization models has been compared. Walters et al. examined WMware, Xen, OpenVZ in terms of different workloads [67]. OpenVZ as an OS-level virtualization solution demonstrated lowest overhead and highest performance. Che et al. [69] also evaluated performance of OpenVZ, Xen, and KVM as implementations of different virtualization models. OpenVZ (OS-based virtualization) had best performance, and KVM (full virtualization) had lower performance than Xen (paravirtualization).

Also, full- and paravirtualization models have been compared. Younge et al. compared Xen, KVM, VirtualBox, and WMware [70]. KVM had best overall performance followed by VirtualBox (full virtualization). Full- and paravirtualization models with Xen has

been studied [71], which indicated that overhead of full virtualization is at least 35% larger, when compared to paravirtualization. Li [72] compared a commercial hypervisor, Xen, and KVM with different Hadoop-based benchmarks. Biggest performance differences were observed with I/O bound benchmarks, while CPU-bound benchmarks revealed smaller differences between hypervisors. HW-assisted virtualization was tested with Xen and KVM by Palit et al. [73]. Particularly, PCI passthrough was used to enable direct access to hardware devices by the VMs. The results indicated near-native performance, and Xen achieved better overall results, when compared to KVM.

Finally, paravirtualization and OS-based virtualization have been compared. Padala et al. compared Xen and OpenVZ in different configurations [74]. The results indicated that Xen had larger overhead, which resulted in higher performance for OpenVZ. Different OS-level/container-based virtualization solutions have been compared to Xen by Xavier et al. [68]. In overall, Xen achieved lower performance than the container-based virtualization solutions. LXC achieved best performance among the container-based solutions. However, isolation of performance between VMs was best with Xen, which may be a disadvantage of OS-based virtualization solutions.

Cloud computing has been characterized by dynamic provisioning of computing resources based on service-level agreements between service provider and consumer [75]. Cloud computing is an umbrella concept, which covers also virtualization solutions. Many open source implementations have been developed for cloud computing [76]. Dukaric and Juric [77] presented a taxonomy and architecture for cloud solutions, and provided a mapping between the framework and different open source cloud implementations.

Architectural and philosophical differences and similarities between Eucalyptus, OpenNebula, and Nimbus cloud solutions were compared by Sempolinski [78]. Wen et al. [79] compared OpenNebula and OpenStack in terms of architecture, support for virtualization hypervisors, cloud APIs, and security aspects. OpenStack, OpenNebula, Nimbus and Eucalyptus were compared in terms of interfaces, hypervisors, networking, deployment, and storage in order to evaluate suitability for FutureGrid testing environment [80]. Also, scalability of provisioning physical and virtual machines was tested, and OpenStack was observed to achieve the best performance. Huang et al. [81] compared CloudStack, Eucalyptus, and OpenNebula for geoscience applications. Especially, the implementations were compared in terms of features and performance (CPU, I/O, memory hierarchy, network transfer, and geoscience applications). A difference was observed in support for web applications, where OpenNebula had slightly better performance, because traffic is not routed via a cloud controller in the architecture. OpenNebula also achieved best performance in geoscience applications, although with a small margin, when compared to the alternatives. Huang et al. also compared afore-mentioned solutions against the performance of a bare-metal cluster [82]. The results indicate that cloud solutions have ∼10% overhead from virtualization and management, which increases when more virtual machines are used. OpenNebula achieved best performance, when compared to the other cloud solutions (CloudStack, Eucalyptus).

5.2.6. New technology frameworks

In addition, completely new analytics frameworks have been developed. Borkar et al. present ASTERIX as a new approach, which provides an alternative to the Hadoop-based data processing paradigm and architecture [83,90]. A new parallel, semi-structured information management system has been created, which consists of three architectural layers. The bottom layer is called Hyracks, which is a data-intensive runtime. The top layer is a full parallel DBMS with a flexible data model and a query language. The middle layer is Algebricks, which is a virtual machine for parallel query processing and optimization. Performance of the approach has been extensively studied, which indicated better results in comparison to Hadoop-based systems.

Flink/Stratosphere follows a similar approach as ASTERIX, and is currently being developed in Europe [84]. It has two layers. PACT is a new programming model, and Nephele is a parallel execution engine. Nephele is the lower layer, and executes data flow graphs. PACT is a generalization of the MapReduce programming model, but also includes a richer set of operators. It offers Java and Scala APIs for developers. Flink applies in-memory data transfers and data pipelining for increasing of performance.

Flink use cases have been published for illustrating the benefits of the framework. Nephele has been extended for detecting violations in user-defined QoS constraints, and optimization of job execution without user intervention [85]. The approach improved processing latency in a video stream augmentation task significantly, when compared to alternatives (no optimization, Hadoop Online [34]). Flink has also been utilized for data mining of users in a social media service [86]. Specifically, PACT programming, execution of data flows, and visualization of analysis results has been demonstrated [86].

5.2.7. Commercial services

Also, many commercial infrastructure services are available for realisation of big data systems (see Appendix C (Table C.1) for references). The classification of commercial services in Fig. 16 was inspired by Feinleib [87]. Software-as-a-Service (SaaS) companies have specialized in the collection and analytics of big data. Examples include Flurry or Medio, which have specialized in data collection and analytics for mobile applications and devices. End users apply a Software Development Kit (SDK), which is obtained from the SaaS-provider for data collection from mobile devices. A user interface is provided for end users for getting access to data metrics. Jaspersoft BI for AWS is another SaaS-based solution for execution of BI based on Amazon's infrastructure and services.

Infochimps provides Infrastructure-as-a-Service (IaaS) for data collection and analysis in cloud-based deployments. Especially, public interfaces are provided for controlling data collection and analysis processes. Also, different virtual cloud networking solutions are available for execution of the service. Amazon EC2 and Windows Azure are other main IaaS solutions.

Data-as-a-Service (DaaS) providers offer a public API for getting access to data, which has been collected from various sources. The extracted data is typically context related. Factual provides an API for getting access to location and context-based data, which has been collected from Factual's partners. Factual's clients can use data and other provided tools for building of context-based mobile applications. Another example is Gnip, which collects data from various social media sources (e.g. Twitter), and provides different services for getting access to it.

A Big data system may also be built on company's enterprise infrastructure. Cloudera is an example of an "Analytics infrastructure", which provides a Hadoop-based platform for execution of data analytics jobs in enterprise environment. Jaspersoft BI Enterprise is an analytics platform for enterprise domain, and consists of several functionalities (ETL, OLAP, in-memory server, visualization). 10Gen provides "Operational infrastructure" for enterprise grade databases and management based on MongoDB technology. BigMemory from Terracotta enables a distributed in-memory data management solution.

Log services enable collection, management, and analysis of logging data collected from various sources. Splunk enables organizations to collect and analyse machine generated log data. Splunk engine can be deployed in enterprise or virtual environments. Log-

gly applies an alternative approach by executing data collection of logs in the cloud environment, and provides a Representational State Transfer (REST) interface for log transmission. A web interface is used for getting insights to analysed data.

Additionally, many advertisement, marketing and media solutions utilizing big data are available. Example companies include Collective and RocketFuel.

6. Analysis

6.1. Reference architecture

A detailed and high level view of the reference architecture (Figs. A.1 and 1) for big data systems was designed inductively based on published material of the big data use cases. The reference architecture for big data systems is comprised of semi-detailed functional components [12] and data stores, and data flows between them (research question 1). The presented design contained description of the architectural elements (Section 4.1). The high level view (Fig. 1) contains the same elements as the detailed view (Fig. A.1).

The empirical data enabled mainly design at the level of abstraction, where data flows, data stores, and functionalities were identified. Additionally, the design could include other abstraction levels e.g. software interfaces, which should be part of a facilitation type of architecture [12]. However, the publications did not expose details of related technologies or functionalities, which made architecture design with other building blocks difficult.

Visualization functionality has been defined with several elements due to the heterogeneity of visualization applications. End user clients (e.g. Twitter client app, Netflix app), BI visualization tools (e.g. Microstrategy UI) or simple Dashboarding UI displaying KPIs (BlockMon) may be utilized. Also, Job and Model specification may be realised together in the same user interface, which is applied for visualization of analysis results. However, these functionalities were specified as separate entities in the model. Finally, in many use cases (i.e. LinkedIn, Netflix, FIU-Miner) the visualization application for batch processing results was not described in the publication.

The reference architecture was created inductively based on the published big data implementation architectures. Thus, only the observed functional components and data stores are present in the architecture. If the model is extended in the future based on other big data use cases, the model may need to be updated with new functional components or data stores.

6.2. Classification of technologies and services

The classification of technologies and services (Fig. 16) was created as an answer to the second research question. It was created based on technologies encountered in the analysed big data use cases, and additional survey of literature and commercial products/services. The survey concentrated on technological areas, products and services, which had not been covered in other literature surveys of big data.

In order for the classification to be more useful, relationship to the reference architecture (Fig. 1) should be understood, which is analysed in the following. First, data in big data use cases may be collected from different sources. When data is collected from a streaming data source, log facilities can be applied. This corresponds with stream extraction in the reference architecture. Streaming data is typically saved into a Temporary stream data store (with log facilities (e.g. Kafka/Scribe)) or into a Raw data store (e.g. distributed file system) for batch processing. When structured data is extracted, data is typically stored into a distributed file system (Temporary data store or Raw data store). Key-value stores were applied in the use cases for storing of analysis results (Analysis results store). Extensible record stores and in-memory stores were used as Stream data stores.

The selection of data processing technologies may depend on real-time processing requirements. Tasks with loose timing requirements can be executed as batch processing jobs (Deep analytics). The jobs for batch-based analysis can be specified with tools in the user interface (Job scheduling/specification). Additionally, machine learning tools may be applied for increasing the quality of analysis. Tasks with real-time requirements may be analysed with stream processing tools.

The analysis results may be transformed for OLAP queries (OLAP processing), stored in a database (Analysis results store, Stream analysis results store, Publish & Subscribe store) or visualized (Dashboarding app, End user app). Commercial BI tools may be utilized for visualization of analysis results (Visualization application).

Also, other technologies are related to the realisation of big data systems. Benchmarking tools can be used for system testing, and simulation of different data sources. Novel technology frameworks have been developed as an alternative to data collection and analysis technologies (e.g. Hadoop/MapReduce based processing frameworks). Virtualization technologies and cloud-based solutions may be utilized, when a platform for data collection and analysis is built into a computer cluster.

Commercial services may have large implications from the perspective of domain, which is applied for execution of the functional components. For example, IaaS, SaaS, and DaaS type of services would realise significant portion of the functionality in the service provider's domain. Also, hybrid approaches may be possible. As an example, log data may be generated in the enterprise domain, which is collected and analysed by a Log service provider, and an application would visualize analysis results in the enterprise domain. Finally, Analytics and Operational infrastructure may be considered as an implementation of complete big data reference architecture, which is deployed at the customer's domain of choice.

7. Discussion

The related contributions on big data reference architecture were either on a very high level [9,8] or have focused on a subset of system functionality [19,20]. However, the results of this work can be compared to the reference architecture developed by Meier [7]. We used inductive reasoning in the construction of the reference architecture, while Meier applied a deductive approach. Thus, we differentiated from Meier, and also used more material (seven big data use cases) in the design process. Our model has many similar elements, when compared to Meier's detailed functional model [7]. As differentiated functional components we presented Job specification, Job scheduling, Model specification, Machine learning, Combining, Cleaning, Visualization and Compression functionalities. Also, Jobs and Models data stores were described. Some of the functionalities and stores of Meier's reference architecture are not present in our model, because we did not observe these elements in the studied use cases. However, many elements between the models are similar (e.g. Raw data store corresponds to Raw data storage in Meier's work). In overall, we achieved a similar, but differentiated model, when compared to the model presented by Meier [7]. Other researchers may also replicate and differentiate our research, as the source material is publicly available.

Also, SOLID architecture for big data can be compared to our work [10]. SOLID consists of an online layer capturing incoming data to the system, merge layer consisting of batch processing, data layer as the main storage, index layer providing fast access to data,

and service layer providing an interface to end users. The main difference is that separate stream and batch processing layers are not considered in SOLID. Also, many of the elements are absent, which we encountered in the realised use cases. However, there is correspondence between the models. Online layer in SOLID corresponds to Extraction of data into Temporary data store, merge layer can be considered as Deep analytics in our model, data layer is associated with the different data stores in our model (e.g. Raw data store, Enterprise data store), and service layer may be associated with Serving data stores providing access to visualization.

Our work can also be compared to similar surveys in the area of big data. Chen et al. [14] presented a comprehensive survey on big data, but they did not focus specifically on architectural issues or explicit classification of technologies and commercial products/services. Chen and Zhang's survey on big data overlaps with our work by covering streaming and visualization technologies [11]. They focused less on classification of technologies and services, which was covered in our work. Furthermore, they identified architecture as one of the top priorities in the design of big data systems, which was the focus in our work. Begoli's [13] state-of-the-art survey contains many technologies, and architectural taxonomies, which are also covered in this document. However, the survey is shorter and has a narrower focus. The big data mining framework by Wu et al. [15] has a higher level abstraction, and a different focus (data mining), when compared to our work. Cuzzocrea et al. [16] discussed OLAP over big data, big data posting and privacy as part of big data research agenda. These aspects may be considered as complementary to our focus on other aspects of big data research. Westerlund et al. suggested a generalized architecture for predictive analytics [89]. The proposal is mainly focused on generation of jobs for machine learning to be executed in a server component, which corresponds to a subset of functionality in our reference architecture.

A limitation of the proposed classification is concentration on selected technologies in the survey. However, other authors have covered other technological topics in earlier surveys: batch processing [11,15,14], machine learning [11], data mining [15, 11], storage systems [14], statistical tools [14,11], and document-oriented databases [14].

Another limitation of this work is that the reference architecture should be evaluated with a real big data use case, which would complete step 6 of the research method (Appendix A). However, the model was created inductively based on the realised use cases. Thus, the final step (6) [17] in the design of a practice-driven reference architecture should be less critical, when compared to design based on a research-driven approach [17], which is not based on architecture implementations.

8. Conclusion

This paper concentrated on reference architecture, technologies, and commercial products/services for big data systems. The aim of the work is to facilitate architecture design, and technology or commercial product/service selection in the construction of big data systems. First, related work on big data surveys, reference architectures, and use cases were presented. The main research question dealt with elements contained in big data reference architecture. As an answer, reference architecture was designed, which consists of functional components, data stores, and connecting data flows. The architecture was constructed with inductive reasoning based on empirical data from the published big data use cases. Also, a mapping between the architecture model and the use cases was presented. Subsequently, related work in big data area was surveyed. Especially, the survey focused on stream processing, graph modelling, business intelligence and visualization technologies, big data benchmarks, virtualization and cloud solutions, new technology frameworks, and commercial products/services. The second research question focused on classification of technologies and products/services for big data systems. A classification was presented as an answer, which included material from the reviewed big data use cases and related work.

Acknowledgements

The authors acknowledge Markus Meier for design of the reference architecture for big data systems, which inspired the authors for replication and differentiation in this work. Additionally, we acknowledge developers and publishers of the reviewed big data use cases, whose publications were used as empirical material in this study.

This research has been carried out in Digile Need for Speed program, and it has been partially funded by Tekes (the Finnish Funding Agency for Technology and Innovation). The research was also partially conducted in ICARE (Innovative Cloud Architecture for Entertainment) project, which is funded by Tekes (ITEA3 program).

Appendix A. Detailed view of the reference architecture

Fig. A.1 presents a detailed view of the reference architecture. Particularly, functionalities, data stores, and data flows of the use cases have been mapped to the architecture.

Appendix B. Methods

This section describes the research method for construction of the reference architecture. We followed replication study approach for research [88]. First, we replicated Meier's approach in the design of the reference architecture for big data systems (differentiated replication [88]). Second, we differentiated [88] from the original approach by using more big data use cases in the design of the reference architecture, and by utilizing inductive reasoning in the design. In the following the differentiated research approach has been described:

It has been defined that an empirically-grounded architecture should be created by following a step-wise approach [17]. The procedure consists of defining a type for the reference architecture (step 1), selecting of a design strategy (step 2), empirical acquisition of data (step 3), construction of reference architecture (step 4), enabling variability (step 5), and evaluation of the reference architecture (step 6). Initially (step 1), the type of reference architecture can be defined based on the framework by Angelov et al. [12], which covers goal and context for a reference architecture. A facilitation type of reference architecture (type 3 in [12]) was chosen, which is designed by an independent organization for utilization by multiple organizations. The main components of the reference architecture are described in an abstract or semi-detailed level.

Subsequently (step 2), design strategy was selected to be practise-driven i.e. architecture was designed based on the existing big data architectures/realisations. Empirical data was gathered (step 3) from publications and blogs by developers of the big data use cases for construction of the architecture. The original research by Meier was differentiated by using material from seven big data use cases (Meier used two similar use cases). We replicated construction of the reference architecture, which was differentiated by consideration of the additional use cases (step 4). Additionally, we constructed the reference architecture inductively based on the realised use cases, whereas Meier applied a deductive approach. Especially, functionalities and data stores in the use cases were analysed, and generalized into one common reference architecture. Variability was not defined to the architecture (step 5). Evaluation of the constructed architecture with a real big data use case is left for future work (step 6).

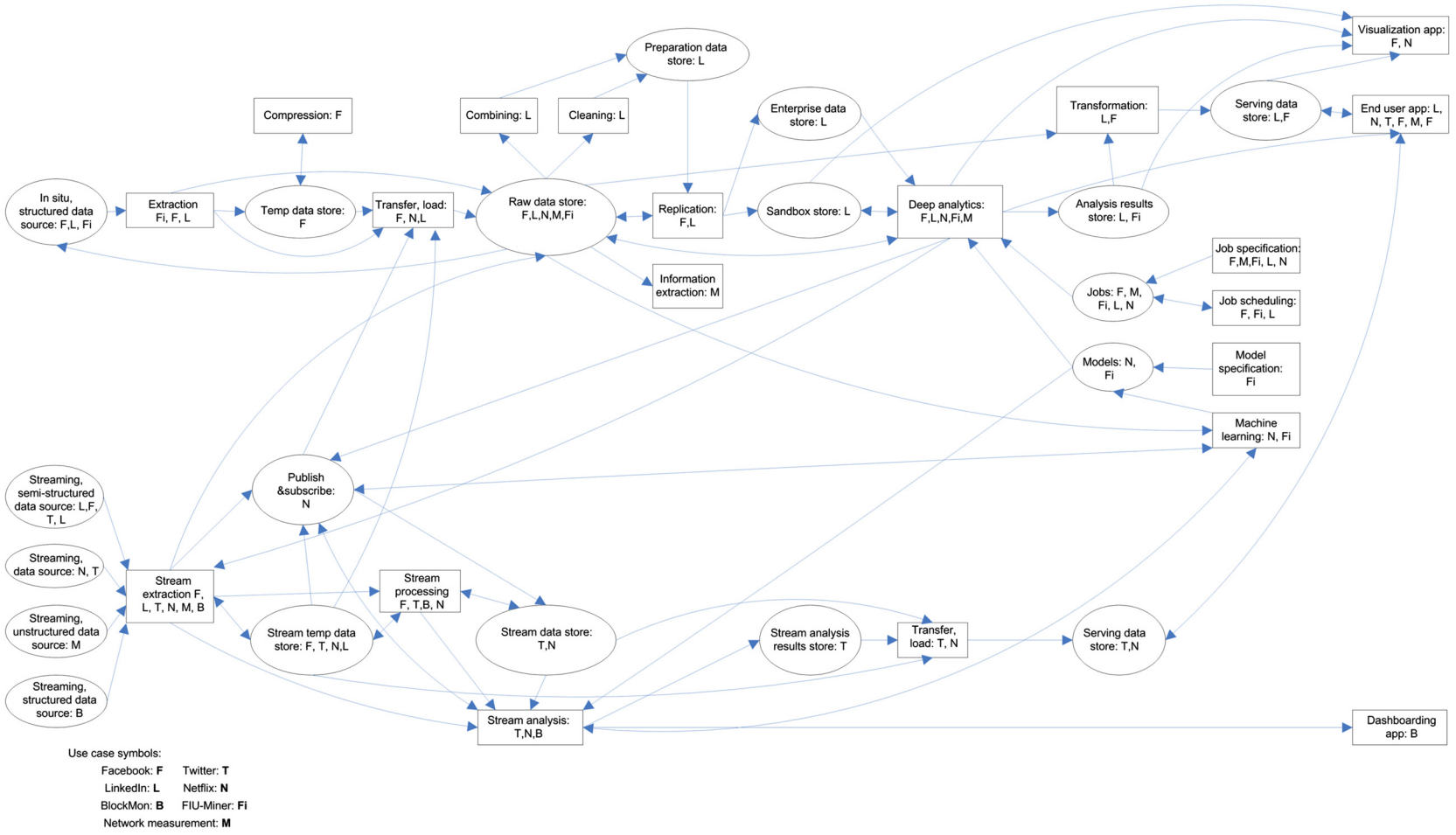

Fig. A.1. Detailed design of the reference architecture. Data stores are depicted as ellipsis, functionalities as rectangles, and data flows as arrows. Letter symbols describe mapping between architectural concepts and use cases.

Appendix C. References to commercial products and services

Table C.1
References to commercial solutions and services.

Product/company	Reference
Amazon EC2	http://aws.amazon.com/ec2/
AMD-V	http://www.amd.com/us/solutions/servers/virtualization/Pages/virtualization.aspx
BigMemory	http://terracotta.org/products/bigmemory
Cloudera	http://www.cloudera.com/
Collective	http://collective.com/
Datameer	http://www.datameer.com/
Factual	http://www.factual.com/
Flurry	http://www.flurry.com/
GNIP	http://gnip.com/
Infochimps	http://www.infochimps.com/
Intel-VT	http://ark.intel.com/products/virtualizationtechnology
Jaspersoft	http://www.jaspersoft.com/
Karmasphere	http://www.karmasphere.com/
Loggly	https://www.loggly.com/
Medio	http://medio.com/
Platfora	http://www.platfora.com/
QlikView	http://www.qlik.com/fi-fi
RocketFuel	http://rocketfuel.com/
SAS Visual Analytics	http://www.sas.com/
Splunk	http://www.splunk.com/
Spotfire	http://spotfire.tibco.com/
SQLStream	http://www.sqlstream.com/
Streambase	www.streambase.com/
Tableau	http://www.tableausoftware.com/
Windows Azure	https://www.windowsazure.com/
WMware	http://www.vmware.com/
10Gen	http://www.mongodb.com/

References

[1] R. Sumbaly, J. Kreps, S. Shah, The "Big Data" Ecosystem at LinkedIn, in: 2013 ACM SIGMOD International Conference on Management of Data, New York, New York, USA, 22–27 June, 2013.

[2] X. Amatriain, Big & Personal: data and models behind Netflix recommendations, in: The 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, Chicago, Illinois, USA, 11 August, 2013.

[3] G. Mishne, Fast data in the era of big data: Twitter's real-time related query suggestion architecture, in: The 2013 ACM SIGMOD International Conference on Management of Data, New York, New York, USA, 22–27 June, 2013.

[4] D. Simoncelli, M. Dusi, F. Gringoli, S. Niccolini, Stream-monitoring with BlockMon: convergence of network measurements and data analytics platforms, ACM SIGCOMM Commun. Rev. 43 (2013) 29–35.

[5] C. Zeng, et al., FIU-miner: a fast, integrated, and user-friendly system for data mining in distributed environment, in: 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, 11–14 August, 2013.

[6] A. Thusoo, et al., Data warehousing and analytics infrastructure at Facebook, in: 2010 ACM SIGMOD International Conference on Management of Data, Indianapolis, Indiana, USA, 6–11 June, 2010.

[7] M. Meier, Towards a big data reference architecture, Master's thesis, Eindhoven University of Technology, October 2013.

[8] R. Schmidt, M. Möhring, Strategic alignment of cloud-based architectures for big data, in: 17th IEEE International Enterprise Distributed Object Computing Conference Workshops, Vancouver, Canada, 9–13 September, 2013.

[9] Y. Demchenko, C. Ngo, P. Membrey, Architecture framework and components for the Big Data Ecosystem, SNE Technical Report, University of Amsterdam, September 12, 2013.

[10] C.E. Cuesta, M.A. Martinez-Prieto, J.D. Fernandez, Towards an architecture for managing big semantic data in real-time, in: 7th European Conference on Software Architecture, Montpellier, France, 1–5 July, 2013.

[11] C.L.P. Chen, C. Zhang, Data-intensive applications, challenges, techniques and technologies: a survey on Big Data, Inf. Sci. 275 (2014) 314–347.

[12] A. Angelov, P. Grefen, D. Greefhorst, A framework for analysis and design of software reference architectures, Inf. Softw. Technol. 54 (2012) 417–431.

[13] E. Begoli, A short survey on the state of the art in architectures and platforms for large scale data analysis and knowledge discovery from data, in: The 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA), Helsinki, Finland, 20–24 August, 2012.

[14] M. Chen, S. Mao, Y. Liu, Big data: a survey, Mob. Netw. Appl. 18 (2014).

[15] X. Wu, G. Wu, W. Ding, Data mining with big data, IEEE Trans. Knowl. Data Eng. 28 (2014) 97–106.

[16] A. Cuzzocrea, D. Sacca, J.D. Ullman, Big data: a research agenda, in: The 17th International Database Engineering & Applications Symposium, Barcelona, Spain, 09–11 October, 2013.

[17] M. Galster, P. Avgeriou, Empirically-grounded reference architectures: a proposal, in: Joint ACM SIGSOFT Conference on Quality of Software Architectures and ACM SIGSOFT Conference on Quality of Software Architectures and ACM SIGSOFT Symposium on Architecting Critical Systems, Boulder, Colorado, USA, June 20–24, 2011.

[18] A. Zimmermann, et al., Towards and integrated service-oriented reference enterprise architecture, in: International Workshop on Ecosystem Architectures, Saint Petersburg, Russia, 19 August, 2013.

[19] K.A. Doshi, et al., Blending SQL and NewSQL approaches reference architectures for enterprise big data challenges, in: The International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, Beijing, China, 10–12 October, 2013.

[20] T. Zhong, et al., On mixing high-speed updates and in-memory queries a big-data architecture for real-time analytics, in: IEEE International Conference on Big Data, Santa Clara, California, USA, 6–9 October, 2013.

[21] J. Lin, D. Ryaboy, Scaling big data mining infrastructure: the Twitter experience, ACM SIGKDD Explor. Newsl. 14 (2013) 6–19.

[22] G.L. Lee, J. Lin, C. Liu, A. Lorek, D. Ryaboy, The unified logging infrastructure for data analytics at Twitter, in: The 38th International Conference on Very Large Databases, Istanbul, Turkey, 27–31 August, 2012.

[23] P. Gupta, et al., WTF: the who to follow service at Twitter, in: The International World Wide Web Conference, Rio de Janeiro, Brazil, 13–17 May, 2013.

[24] M. Dusi, et al., BlockMon: flexible and high-performance big data stream analytics platform and its use cases, NEC Tech. J. 7 (2012) 102–106.

[25] Y. Lee, Y. Lee, Toward scalable internet traffic measurement and analysis with hadoop, ACM SIGCOMM Commun. Rev. 43 (2013) 5–13.

[26] Y. Lee, W. Kang, Y. Lee, A hadoop-based packet trace processing tool, in: International Workshop on Traffic Monitoring and Analysis, Vienna, Austria, April 27, 2011.

[27] J. Kreps, N. Narkhede, J. Rao, Kafka: a distributed messaging system for log processing, in: The 6th International Workshop on Networking Meets Databases, Athens, Greece, 12 June, 2011.

[28] L. Wu, et al., Avatara: OLAP for web-scale analytics products, in: 38th International Conference on Very Large Databases, Istanbul, Turkey, 27–31 August, 2012.

[29] M. Busch, et al., EarlyBird: real-time search at Twitter, in: 2012 IEEE 28th International Conference on Data Engineering, Washington, DC, USA, 1–5 April, 2012.

[30] X. Amatriain, System architectures for personalized recommendations, Available via Netflix, http://techblog.netflix.com/2013/03/system-architectures-for.html, accessed 25 Nov., 2013.

[31] J. Boulon, et al., Chukwa: a large-scale monitoring system, in: Cloud Computing and its Applications, Chicago, Illinois, USA, 22–23 October, 2008.

[32] A. Woznica, R. Nguyen, A. Kalousis, Model mining for robust feature selection, in: The 18th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Beijing, China, 12–16 August, 2012.

[33] M. Mavani, L. Ragha, MapReduce frame work: investigating suitability for faster data analytics, Adv. Comput. Commun. Control CCIS 361 (2013) 119–130.

[34] T. Condie, et al., MapReduce online, in: The 7th USENIX Conference on Networked Systems Design and Implementation, San Jose, California, USA, 28–30 April, 2010.

[35] W. Lam, et al., Muppet: MapReduce-style processing of fast data, in: The 38th International Conference on Very Large Databases, Istanbul, Turkey, 27–31 August, 2012.

[36] L. Neumeyer, B. Robbins, A. Nair, A. Kesari, S4: distributed stream computing platform, in: The IEEE International Conference on Data Mining Workshops, 13 December, 2010.

[37] J. Chauhan, S.A. Chowdhury, D. Makaroff, Performance evaluation of Yahoo! S4: a first look, in: The 7th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Victoria, Canada, 12–14 November, 2012.

[38] SQLStream, processing and analysing streams of CDRs in real time, http://www.sqlstream.com/wp-content/uploads/2014/02/SQLstream_Whitepaper_CDR-Analytics-update.pdf, accessed 13/04/2014.

[39] Z. Qian, et al., TimeStream: reliable stream computation in the cloud, in: Eurosys Conference, Prague, Czech Republic, 14–17 April, 2013.

[40] M. Zaharia, et al., Discretized streams: fault-tolerant streaming computation at scale, in: 24th ACM Symposium on Operating System Principles, Farmington, Pennsylvania, USA, 3–8 November, 2013.

[41] D.G. Murray, et al., Naiad: a timely dataflow system, in: The 24th ACM Symposium on Operating System Principles, Farmington, Pennsylvania, USA, 3–6 November, 2013.

[42] H. Matsuura, M. Ganse, T. Suzumura, A highly efficient consolidated platform for stream computing and hadoop, in: The 26th International Parallel and Distributed Processing Symposium, Shanghai, China, 21–25 May, 2012.

[43] R. Angles, C. Gutierrez, Survey of graph database models, ACM Comput. Surv. 40 (2008) 1–39.
[44] M. Stonebraker, S. Madden, P. Dubey, Intel "Big Data" science and technology center vision and execution plan, SIGMOD Rec. 42 (2013) 44–49.
[45] R. Angles, A comparison of current graph database models, in: The 28th IEEE International Conference on Data Engineering Workshops, Arlington, Virginia, USA, 1–5 April, 2012.
[46] B. Shao, H. Wang, Y. Xiao, Managing and mining large graphs: systems and implementations, in: SIGMOD 2012, Scottsdale, Arizona, USA, 20–24 May, 2012.
[47] M. Dayarathna, T. Suzumura, Graph database benchmarking on cloud environments with XGDBench, Autom. Softw. Eng. 21 (2013).
[48] R. Angles, A. Prat-Perez, D. Dominguez-Sal, J. Larriba-Pey, Benchmarking database systems for social network applications, in: The 1st International Workshop on Graph Data Management Experience and Systems, New York, USA, June 23, 2013.
[49] D.D. Abreu, et al., Choosing between graph databases and RDF engines for consuming and mining linked data, in: 4th International Workshop on Consuming Linked Data, Sydney, Australia, 22 October, 2013.
[50] J. Huang, D.J. Abadi, K. Ren, Scalable SPARQL querying of large graphs, in: The 37th International Conference on Very Large Data Bases, Seattle, Washington, USA, 29 August–3 September, 2011.
[51] K. Zeng, et al., A distributed graph engine for web scale RDF data, in: 39th International Conference on Very Large Databases, Riva del Garda, Trento, Italy, 26–30 August, 2013.
[52] B. Shao, H. Wang, Y. Li, Trinity: a distributed graph engine on a memory cloud, in: 2013 ACM SIGMOD International Conference on Management of Data, New York, New York, USA, 22–27 June, 2013.
[53] G. Malewicz, et al., Pregel: a system for large-scale graph processing, in: SIGMOD 2010, Indianapolis, Indiana, USA, 6–11 June, 2010.
[54] Y. Low, et al., Distributed GraphLab: a framework for machine learning and data mining in the cloud, in: The 38th International Conference on Very Large Databases, Istanbul, Turkey, 27–31 August, 2012.
[55] A. Kyrölä, G. Blelloch, C. Guestrin, in: Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation, Hollywood, California, 8–10 October, 2012.
[56] S. Chaudhuri, U. Dayal, V. Narasayya, An overview of business intelligence technology, Commun. ACM 54 (2011) 88–98.
[57] J. Choo, H. Park, Customizing computational methods for visual analytics with big data, IEEE Comput. Graph. Appl. 33 (2013) 22–28.
[58] J. Beyer, et al., Exploring the connectome petascale volume visualization of microscopy data streams, IEEE Comput. Graph. Appl. 33 (2013) 50–61.
[59] Y. Chen, S. Alspaugh, R. Katz, Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads, in: The 38th International Conference on Very Large Databases (VLDB'12), Istanbul, Turkey, 27–31 August, 2012.
[60] X. Qin, X. Zhou, A survey on benchmarks for big data and some more considerations, in: 14th International Conference on Intelligent Data Engineering and Automated Learning, Hefei, Anhui, China, 20–23 October, 2013.
[61] L. Wang, et al., BigDataBench: a big data benchmark suite from internet services, in: 20th IEEE International Symposium on High Performance Computer Architecture (HPCA-2014), Orlando, Florida, USA, 15–19 February, 2014.
[62] A. Ghazal, et al., BigBench: towards an industry standard benchmark for big data analytics, in: SIGMOD 2013, New York, USA, 22–27 June, 2013.
[63] T. Rabl, H. Jacobsen, Big data generation, in: Workshop on Big Data Benchmarking, San Diego, California, USA, 8–9 May, 2012.
[64] C. Baru, et al., Benchmarking big data systems and the BigData Top100 list, Big Data J. 1 (2013).
[65] T.G. Armstrong, V. Ponnekanti, D. Borthakur, LinkBench: a database benchmark based on the Facebook social graph, in: SIGMOD 2013, New York, USA, 22–27 June, 2013.
[66] J. Quan, Y. Shi, M. Zhao, W. Yang, The implications from benchmarking three big data systems, in: IEEE International Conference on Big Data, Santa Clara, California, USA, 6–9 October, 2013.
[67] J.P. Walters, et al., A comparison of virtualization technologies for HPC, in: 22nd International Conference on Advanced Information Networking and Applications, Ginowan, Okinawa, Japan, 25–28 March, 2008.
[68] M.G. Xavier, Performance evaluation of container-based virtualization for high performance computing environments, in: 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Belfast, Northern Ireland, 27 February–1 March, 2013.
[69] J. Che, et al., A synthetical performance evaluation of OpenVZ, Xen and KVM, in: The IEEE Asia-Pacific Services Computing Conference, Hangzhou, China, 6–10 December, 2010.
[70] A.J. Younge, et al., Analysis of virtualization technologies for high performance computing environments, in: IEEE 4th International Conference on Cloud Computing, Washington, DC, USA, 4–9 July, 2011.
[71] H. Fayyad-Kazan, L. Perneel, M. Timmerman, Full and para-virtualization with Xen: a performance comparison, J. Emerg. Trends Comput. Infor. Sci. 4 (2013) 719–727.
[72] J. Li, et al., Performance overhead among three hypervisors: an experimental study using hadoop benchmarks, in: IEEE International Congress on Big Data, Santa Clara, CA, USA, 6–9 October, 2013.
[73] H.N. Palit, et al., Evaluating hardware-assisted virtualization for deploying HPC-as-a-Service, in: Virtualization Technologies in Distributed Computing, New York, NY, USA, 18 June, 2013.
[74] P. Padala, et al., Performance Evaluation of Virtualization Technologies for Server Consolidation, HP Laboratories Palo Alto, April 11, 2007.
[75] R. Buyya, et al., Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (2009) 599–616.
[76] E. Voras, et al., Evaluating open-source cloud computing solutions, in: 34th International Convention MIPRO 2011, Opatija, Croatia, 23–27 May, 2011.
[77] R. Dukaric, M.B. Juric, Towards a unified taxonomy and architecture of cloud frameworks, Future Gener. Comput. Syst. 29 (2013) 1196–1210.
[78] P. Sempolinski, D. Thain, A comparison and critique of eucalyptus, OpenNebula and Nimbus, in: 2nd IEEE International Conference on Cloud Computing Technology and Science, Indianapolis, Indiana, USA, 30 November–3 December, 2013.
[79] X. Wen, et al., Comparison of open-source cloud management platforms: OpenStack and OpenNebula, in: 9th International Conference on Fully Systems and Knowledge Discovery, Chongqing, Sichuan, China, 29–31 May, 2012.
[80] G. von Laszewski, et al., Comparison of multiple cloud networks, in: 5th International Conference on Cloud Computing, Honolulu, Hawaii, USA, 24–29 June, 2012.
[81] Q. Huang, et al., Evaluating open source cloud computing solutions for geosciences, Comput. Geosci. 59 (2013) 41–52.
[82] Q. Huang, et al., An experimental study of open-source cloud platforms for dust storm forecasting, in: AMC SIGSPATIAL, Redondo Beach, California, USA, 6–9 November, 2012.
[83] V. Borkar, M.J. Carey, C. Li, Inside "Big Data Management": ogres, onions, or parfaits?, in: 15th International Conference on Extending Database Technology, Berlin, Germany, 27–30 March, 2012.
[84] S. Ewen, Iterative parallel data processing with stratosphere: an inside look, in: SIGMOD 2013, New York, USA, 22–27 June, 2013.
[85] B. Lohrmann, D. Warneke, O. Kao, Nephele streaming: stream processing under QoS constraints at scale, J. Cluster Comput. 16 (2013).
[86] C. Boden, V. Markl, M. Karnstedt, M. Fernandez, Large-scale social-media analytics on stratosphere, in: The International World Wide Web Conference, Rio de Janeiro, Brazil, 13–17 May, 2013.
[87] D. Feinleib, Big data landscape, http://blogs-images.forbes.com/davefeinleib/files/2012/07/Big-Data-Landscape-Jul-4-2012.00111.png, accessed 01/01/2014.
[88] R.M. Lindsay, A.S.C. Ehrenberg, The design of replicated studies, Am. Stat. 47 (1993) 217–228.
[89] M. Westerlund, et al., A generalized scalable software architecture for analyzing temporally structured big data in the cloud, New Perspect. Inform. Syst. Technol. 1 (2014) 559–569.
[90] S. Alsubaiee, et al., AsterixDB: a scalable, open source BDMS, in: The 40th International Conference on Very Large Databases (VLDB'14), Hangzhou, China, 1–5 September, 2014.
[91] S. Abiteboul, Querying semi-structured data, in: International Conference on Database Theory, Delphi, Greece, 8–10 January, 1997.
[92] J. Chen, et al., Big data challenge: a data management perspective, Front. Comput. Sci. 2 (2013) 157–164.