A Software Reference Architecture for Semantic-Aware Big Data Systems

Sergi Nadal[a,*], Victor Herrero[a], Oscar Romero[a], Alberto Abelló[a], Xavier Franch[a], Stijn Vansummeren[b], Danilo Valerio[c]

[a] *Universitat Politècnica de Catalunya - BarcelonaTech*
[b] *Université Libre de Bruxelles*
[c] *Corporate Research and Technology, Siemens AG Österreich*

Abstract

Context: Big Data systems are a class of software systems that ingest, store, process and serve massive amounts of heterogeneous data, from multiple sources. Despite their undisputed impact in current society, their engineering is still in its infancy and companies find it difficult to adopt them due to their inherent complexity. Existing attempts to provide architectural guidelines for their engineering fail to take into account important Big Data characteristics, such as the management, evolution and quality of the data.
Objective: In this paper, we follow software engineering principles to refine the λ-architecture, a reference model for Big Data systems, and use it as seed to create *Bolster*, a software reference architecture (SRA) for semantic-aware Big Data systems.
Method: By including a new layer into the λ-architecture, the Semantic Layer, *Bolster* is capable of handling the most representative Big Data characteristics (i.e., Volume, Velocity, Variety, Variability and Veracity).
Results: We present the successful implementation of *Bolster* in three industrial projects, involving five organizations. The validation results show high level of agreement among practitioners from all organizations with respect to standard quality factors.
Conclusion: As an SRA, *Bolster* allows organizations to design concrete architectures tailored to their specific needs. A distinguishing feature is that it provides *semantic-awareness* in Big Data Systems. These are Big Data system implementations that have components to simplify data definition and exploitation. In particular, they leverage metadata (i.e., data describing data) to enable (partial) automation of data exploitation and to aid the user in their

*Corresponding author. Address: Campus Nord Omega-125, UPC - dept ESSI, C/Jordi Girona Salgado 1-3, 08034 Barcelona, Spain

Email addresses: `snadal@essi.upc.edu` (Sergi Nadal), `vherrero@essi.upc.edu` (Victor Herrero), `oromero@essi.upc.edu` (Oscar Romero), `aabello@essi.upc.edu` (Alberto Abelló), `franch@essi.upc.edu` (Xavier Franch), `stijn.vansummeren@ulb.ac.be` (Stijn Vansummeren), `danilo.valerio@siemens.com` (Danilo Valerio)

decision making processes. This simplification supports the differentiation of responsibilities into cohesive roles enhancing data governance.

Keywords: Big Data, Software Reference Architecture, Semantic-Aware, Data Management, Data Analysis

1. Introduction

Major Big Data players, such as Google or Amazon, have developed large Big Data systems that align their business goals with complex data management and analysis. These companies exemplify an emerging paradigm shift towards data-driven organizations, where data are turned into valuable knowledge that becomes a key asset for their business. In spite of the inherent complexity of these systems, software engineering methods are still not widely adopted in their construction (Gorton and Klein, 2015). Instead, they are currently developed as ad-hoc, complex architectural solutions that blend together several software components (usually coming from open-source projects) according to the system requirements.

An example is the Hadoop ecosystem. In Hadoop, lots of specialized Apache projects co-exist and it is up to Big Data system architects to select and orchestrate some of them to produce the desired result. This scenario, typical from immature technologies, raises high-entry barriers for non-expert players who struggle to deploy their own solutions overwhelmed by the amount of available and overlapping components. Furthermore, the complexity of the solutions currently produced requires an extremely high degree of specialization. The system end-user needs to be what is nowadays called a "data scientist", a data analysis expert proficient in managing data stored in distributed systems to accommodate them to his/her analysis tasks. Thus, s/he needs to master two profiles that are clearly differentiated in traditional Business Intelligence (BI) settings: the data steward and the data analyst, the former responsible of data management and the latter of data analysis. Such combined profile is rare and subsequently entails an increment of costs and knowledge lock-in.

Since the current practice of ad-hoc design when implementing Big Data systems is hence undesirable, improved software engineering approaches specialized for Big Data systems are required. In order to contribute towards this goal, we explore the notion of Software Reference Architecture (SRA) and present *Bolster*, an SRA for Big Data systems. SRAs are generic architectures for a class of software systems (Angelov et al., 2012). They are used as a foundation to derive software architectures adapted to the requirements of a particular organizational context. Therefore, they open the door to effective and efficient production of complex systems. Furthermore, in an emergent class of systems (such as Big Data systems), they make it possible to synthesize in a systematic way a consolidated solution from available knowledge. As a matter of fact, the detailed design of such a complex architecture has already been designated as a major Big Data software engineering research challenge (Madhavji et al., 2015; Esteban, 2016). Well-known examples of SRAs include the successful AUTOSAR SRA (Martínez-Fernández et al., 2015) for the automotive industry, the Internet of Things Architecture (IoT-A) (Weyrich and Ebert, 2016), an SRA for web browsers (Grosskurth and Godfrey, 2005) and the NIST Cloud Computing Reference Architecture (Liu et al., 2012).

As an SRA, *Bolster* paves the road to the prescriptive development of software architectures that lie at the heart of every new Big Data system. Using *Bolster*, the work of the software architect is not to produce a new architecture from a set of independent components that need to be assembled. Instead, the software architect knows beforehand what type of components are needed and how they are interconnected. Therefore, his/her main responsibility is the selection of technologies for those components given the concrete requirements and the goals of the organization. *Bolster* is a step towards the homogeneization and definition of a Big Data Management System (BDMS), as done in the past for Database Management Systems (DBMS) (Garcia-Molina et al., 2009) and Distributed Database Management Systems (DDBMS) (Özsu and Valduriez, 2011). A distinguishing feature of *Bolster* is that it provides an SRA for *semantic-aware* Big Data Systems. These are Big Data system implementations that have components to simplify data definition and data exploitation. In particular, such type of systems leverage on metadata (i.e., data describing data) to enable (partial) automation of data exploitation and to aid the user in their decision making processes. This definition supports the differentiation of responsibilities into cohesive roles, the data steward and the data analyst, enhancing data governance.

Contributions. The main contributions of this paper are as follows:

- Taking as building blocks the five "V's" that define Big Data systems (see Section 2), we define the set of functional requirements sought in each to realize a semantic-aware Big Data architecture. Such requirements will further drive the design of *Bolster*.

- Aiming to study the related work on Big Data architectures, we perform a lightweight Systematic Literature Review. Its main outcome consists on the division of 21 works into two great families of Big Data architectures.

- We present *Bolster*, an SRA for semantic-aware Big Data systems. Combining principles from the two identified families, it succeeds on satisfying all the posed Big Data requirements. *Bolster* relies on the systematic use of semantic annotations to govern its data lifecycle, overcoming the shortcomings present in the studied architectures.

- We propose a framework to simplify the instantiation of *Bolster* to different Big Data ecosystems. For the sake of this paper, we precisely focus on the components of the Apache Hadoop and Amazon Web Services (AWS) ecosystems.

- We detail the deployment of *Bolster* in three different industrial scenarios, showcasing how it adapts to their specific requirements. Furthermore, we provide the results of its validation after interviewing practitioners in such organizations.

Outline. The paper is structured as follows. Section 2 introduces the Big Data dimensions and requirements sought. Section 3 presents the Systematic Literature Review. Sections 4, 5 and 6 detail the elements that compose *Bolster*, an exemplar case study implementing it and the proposed instantiation method respectively. Further, Sections 7 report the industrial deployments and validation. Finally, Section 8 wraps up the main conclusions derived from this work.

2. Big Data Definition and Dimensions

Big Data is a natural evolution of BI, and inherits its ultimate goal of transforming raw data into valuable knowledge. Nevertheless, traditional BI architectures, whose de-facto architectural standard is the Data Warehouse (DW), cannot be reused in Big Data settings. Indeed, the so-popular characterization of Big Data in terms of the three "V's (Volume, Velocity and Variety)" (Jagadish et al., 2014), refers to the inability of DW architectures, which typically rely on relational databases, to deal and adapt to such large, rapidly arriving and heterogeneous amounts of data. To overcome such limitations, Big Data architectures rely on NOSQL (Not Only SQL), co-relational database systems where the core data structure is not the relation (Meijer and Bierman, 2011), as their building blocks. Such systems propose new solutions to address the three V's by (i) distributing data and processing in a cluster (typically of commodity machines) and (ii) by introducing alternative data models. Most NOSQL systems distribute data (i.e., fragment and replicate it) in order to parallelize its processing while exploiting the data locality principle, ideally yielding a close-to-linear scale-up and speed-up (Özsu and Valduriez, 2011). As enunciated by the CAP theorem (Brewer, 2000), distributed NOSQL systems must relax the well-known ACID (Atomicity, Consistency, Isolation, Durability) set of properties and the traditional concept of transaction to cope with large-scale distributed processing. As result, data consistency may be compromised but it enables the creation of fault-tolerant systems able to parallelize complex and time-consuming data processing tasks. Orthogonally, NOSQL systems also focus on new data models to reduce the impedance mismatch (Gray et al., 2005). Graph, key-value or document-based modeling provide the needed flexibility to accommodate dynamic data evolution and overcome the traditional staticity of relational DWs. Such flexibility is many times acknowledged by referring to such systems as schemaless databases. These two premises entailed a complete rethought of the internal structures as well as the means to couple data analytics on top of such systems. Consequently, it also gave rise to the Small and Big Analytics concepts (Stonebraker, 2012), which refer to performing traditional OLAP/-Query&Reporting to gain quick insight into the data sets by means of descriptive analytics (i.e., Small Analytics) and Data Mining/Machine Learning to enable predictive analytics (i.e., Big Analytics) on Big Data systems, respectively.

In the last years, researchers and practitioners have widely extended the three "V's" definition of Big Data as new challenges appear. Among all existing definitions of Big Data, we claim that the real nature of Big Data can be covered by five of those "V's", namely: (a) Volume, (b) Velocity, (c) Variety, (d) Variability and (e) Veracity. Note that, in contrast to other works, we do not consider Value. Considering that any decision support system (DSS) is the result of a tightly coupled collaboration between business and IT (García et al., 2016), Value falls into the business side while the aforementioned dimensions focus on the IT side. In the rest of this paper we refer to the above-mentioned "V's" also as Big Data dimensions.

In this section, we provide insights on each dimension as well as a list of linked requirements that we consider a Big Data architecture should fulfill. Such requirements were obtained in two ways: firstly inspired by reviewing related literature on Big Data requirements (Gani et al., 2016; Agrawal et al., 2011; Russom, 2011; Fox and Chang, 2015; Chen and Zhang, 2014); secondly they were validated and refined by informally discussing with the stakeholders from several industrial Big Data projects (see Section 7) and obtaining their feedback. Finally, a summary of devised requirements for each Big Data dimension is depicted in Table 1. Note that such list does not aim to provide an exhaustive set of requirements for Big Data architectures, but a high-level baseline on the main requirements any Big Data architecture should achieve to support each dimension.

2.1. Volume

Big Data has a tight connection with Volume, which refers to the large amount of digital information produced and stored in these systems, nowadays shifting from terabytes to petabytes (**R1.1**). The most widespread solution for Volume is data distribution and parallel processing, typically using cloud-based technologies. Descriptive analysis (Sharda et al., 2013) (**R1.2**), such as reporting and OLAP, has shown to naturally adapt to distributed data management solutions. However, predictive and prescriptive analysis (**R1.3**) show higher-entry barriers to fit into such distributed solutions (Tsai et al., 2015). Classically, data analysts would dump a fragment of the DW in order to run statistical methods in specialized software, (e.g., R or SAS) (Ordonez, 2010). However, this is clearly unfeasible in the presence of Volume, and thus typical predictive and prescriptive analysis methods must be rethought to run within the distributed infrastructure, exploiting the data locality principle (Özsu and Valduriez, 2011).

2.2. Velocity

Velocity refers to the pace at which data are generated, ingested (i.e., dealt with the arrival of), and processed, usually in the range of milliseconds to seconds. This gave rise to the concept of data stream (Babcock et al., 2002) and creates two main challenges. First, data stream ingestion, which relies on a sliding window buffering model to smooth arrival irregularities (**R2.1**). Second, data stream processing, which relies on linear or sublinear algorithms to provide near real-time analysis (**R2.2**).

2.3. Variety

Variety deals with the heterogeneity of data formats, paying special attention to semi-structured and unstructured external data (e.g., text from social networks, JSON/XML-formatted scrapped data, Internet of Things sensors, etc.) (**R3.1**). Aligned with it, the novel concept of Data Lake has emerged (Terrizzano et al., 2015), a massive repository of data in its original format. Unlike DW that follows a *schema on-write* approach, Data Lake proposes to store data as they are produced without any preprocessing until it is clear how they are going to be analyzed (**R3.2**), following the *load-first model-later* principle. The rationale behind a Data Lake is to store raw data and let the data analyst decide how to cook them. However, the extreme flexibility provided by the Data Lake is also its biggest flaw. The lack of schema prevents the system from knowing what is exactly stored and this burden is left on the data analyst shoulders (**R3.3**). Since loading is not that much of a challenge compared to the data transformations (*data curation*) to be done before exploiting the data, the Data Lake approach has received lots of criticism and the uncontrolled dump of data in the Data Lake is referred to as Data Swamp (Stonebraker, 2014).

2.4. Variability

Variability is concerned with the evolving nature of ingested data, and how the system copes with such changes for data integration and exchange. In the relational model, mechanisms to handle evolution of *intension* (**R4.1**) (i.e., schema-based), and *extension* (**R4.2**) (i.e., instance-based) are provided. However, achieving so in Big Data systems entails an additional challenge due to the schemaless nature of NOSQL databases. Moreover, during the lifecycle of a Big Data-based application, data sources may also vary (e.g., including a new social network or because of an outage in a sensor grid). Therefore, mechanisms to handle data source evolution should also be present in a Big Data architecture (**R4.3**).

2.5. Veracity

Veracity has a tight connection with data quality, achieved by means of data governance protocols. Data governance concerns the set of processes and decisions to be made in order to provide an effective management of the data assets (Khatri and Brown, 2010). This is usually achieved by means of best practices. These can either be defined at the organization level, depicting the business domain knowledge, or at a generic level by data governance initiatives (e.g., Six Sigma (Harry and Schroeder, 2005)). However, such large and heterogeneous amount of data present in Big Data systems begs for the adoption of an automated data governance protocol, which we believe should include, but might not be limited to, the following elements:

- Data provenance (**R5.1**), related to how any piece of data can be tracked to the sources to reproduce its computation for lineage analysis. This requires storing metadata for all performed transformations into a common data model for further study or exchange (e.g., the Open Provenance Model (Moreau et al., 2011)).

- Measurement of data quality (**R5.2**), providing metrics such as accuracy, completeness, soundness and timeliness, among others (Batini et al., 2015). Tagging all data with such adornments prevents analysts from using low quality data that might lead to poor analysis outcomes (e.g., missing values for some data).

- Data liveliness (**R5.3**), leveraging on conversational metadata (Terrizzano et al., 2015) which records when data are used and what is the outcome users experience from it. Contextual analysis techniques (Aufaure, 2013) can leverage such metadata in order to aid the user in future analytical tasks (e.g., query recommendation (Giacometti et al., 2008)).

- Data cleaning (**R5.4**), comprising a set of techniques to enhance data quality like standardization, deduplication, error localization or schema matching. Usually such activities are part of the preprocessing phase, however they can be introduced along the complete lifecycle. The degree of automation obtained here will vary depending on the required user interaction, for instance any entity resolution or profiling activity will infer better if user aided.

Including the aforementioned automated data governance elements into an architecture is a challenge, as they should not be intrusive. First, they should be transparent to developers and run as under the hood processes. Second, they should not overburden the overall system performance (e.g., (Interlandi et al., 2015) shows how automatic data provenance support entails a 30% overhead on performance).

2.6. Summary

The discussion above shows that current BI architectures (i.e., relying on RDMS), cannot be reused in Big Data scenarios. Such modern DSS must adopt NOSQL tools to overcome the issues posed by Volume, Velocity and Variety. However, as discussed for Variability and Veracity, NOSQL does not satisfy key requirements that should be present in a mature DSS. Thus, *Bolster* is designed to completely satisfy the aforementioned set of requirements, summarized in Table 1.

3. Related Work

In this section, we follow the principles and guidelines of Systematic Literature Reviews (SLR) as established in (Kitchenham and Charters, 2007). The purpose of this review is to systematically analyze the current landscape of Big Data

Requirement	
1.	*Volume*
R1.1	The BDA shall provide scalable storage of massive data sets.
R1.2	The BDA shall be capable of supporting descriptive analytics.
R1.3	The BDA shall be capable of supporting predictive and prescriptive analytics.
2.	*Velocity*
R2.1	The BDA shall be capable of ingesting multiple, continuous, rapid, time varying data streams.
R2.2	The BDA shall be capable of processing data in a (near) real-time manner.
3.	*Variety*
R3.1	The BDA shall support ingestion of raw data (structured, semi-structured and unstructured).
R3.2	The BDA shall support storage of raw data (structured, semi-structured and unstructured).
R3.3	The BDA shall provide mechanisms to handle machine-readable schemas for all present data.
4.	*Variability*
R4.1	The BDA shall provide adaptation mechanisms to schema evolution.
R4.2	The BDA shall provide adaptation mechanisms to data evolution.
R4.3	The BDA shall provide mechanisms for automatic inclusion of new data sources.
5.	*Veracity*
R5.1	The BDA shall provide mechanisms for data provenance.
R5.2	The BDA shall provide mechanisms to measure data quality.
R5.3	The BDA shall provide mechanisms for tracing data liveliness.
R5.4	The BDA shall provide mechanisms for managing data cleaning.

Table 1: Requirements for a Big Data Architecture (BDA)

architectures, with the goal to identify how they meet the devised requirements, and thus aid in the design of an SRA. Nonetheless, in this paper we do not aim to perform an exhaustive review, but to depict, in a systematic manner, an overview on the landscape of Big Data architectures. To this end, we perform a lightweight SLR, where we focus on high quality works and evaluate them with respect to the previously devised requirements.

3.1. Selection of papers

The search was ranged from 2010 to 2016, as the first works on Big Data architectures appeared by then. The search engine selected was Scopus[1], as it indexes all journals with a JCR impact factor, as well as the most relevant conferences based on the CORE index[2]. We have searched papers with title, abstract or keywords matching the terms "big data" AND "architecture". The list was further refined by selecting papers only in the "Computer Science" and "Engineering" subject areas and only documents in English. Finally, only conference papers, articles, book chapters and books were selected.

By applying the search protocol we obtained 1681 papers covering the search criteria. After a filter by title, 116 papers were kept. We further applied a filter by abstract in order to specifically remove works describing middlewares as part of a Big Data architecture (e.g., distributed storage or data stream management systems). This phase resulted in 44 selected papers. Finally, after reading them, sixteen papers were considered relevant to be included in this section. Furthermore, five non-indexed works considered grey literature were additionally added to the list, as considered relevant to depict the state of the practice in industry. The process was performed by our research team, and in case of contradictions a meeting was organized in order to reach consensus. Details of the search and filtering process are available at (Nadal et al., 2016).

3.2. Analysis

In the following subsections, we analyze to which extent the selected Big Data architectures fulfill the requirements devised in Section 2. Each architecture is evaluated by checking whether it satisfies a given requirement (✓) or it does not (✗). Results are summarized in Table 2, where we make the distinction between custom architectures and SRAs. For the sake of readability, references to studied papers have been substituted for their position in Table 2.

3.2.1. Requirements on Volume

Most architectures are capable of dealing with storage of massive data sets (**R1.1**). However, we claim those relying on Semantic Web principles (i.e. storing RDF data), [A1,A8] cannot deal with such requirement as they are inherently limited by the storage capabilities of triplestores. Great effort is put on improving such capabilities (Zeng et al., 2013), however no mature scalable solution is available in the W3C recommendations[3]. There is an exception to the previous discussion, as SHMR [A14] stores semantic data on HBase. However, this impacts its analytical capabilities with respect to those offered by triplestores. Oppositely, Liquid [A9] is the only case where no data are stored, offering only real-time support and thus not addressing the Volume dimension of Big Data. Regarding analytical capabilities, most architectures satisfy the descriptive level (**R1.2**) via

[1] `http://www.scopus.com`
[2] `http://www.core.edu.au/conference-portal`
[3] `https://www.w3.org/2001/sw/wiki/Category:Triple_Store`

Custom Architectures	*Volume*			*Velocity*		*Variety*			*Variability*			*Veracity*			
	R1.1	R1.2	R1.3	R2.1	R2.2	R3.1	R3.2	R3.3	R4.1	R4.2	R4.3	R5.1	R5.2	R5.3	R5.4
A1 CQELS (Phuoc et al., 2012)	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	✓	✗	✗	✗	✗
A2 AllJoyn Lambda (Villari et al., 2014)	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
A3 CloudMan (Qanbari et al., 2014)	✓	✓	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
A4 AsterixDB (Alsubaiee et al., 2014)	✓	✓	✗	✓	✗	✓	✗	✓	✓	✓	✓	✓	✗	✗	✗
A5 M3Data (Ionescu et al., 2014)	✓	✓	✓	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓
A6 (Twardowski and Ryzko, 2014)	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
A7 λ-arch. (Marz and Warren, 2015)	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
A8 SOLID (Martínez-Prieto et al., 2015)	✗	✓	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
A9 Liquid (Fernandez et al., 2015)	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗
A10 RADStack (Yang et al., 2015)	✓	✓	✗	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓
A11 (Kroß et al., 2015)	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
A12 HaoLap (Song et al., 2015)	✓	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
A13 (Wang et al., 2015)	✓	✓	✓	✗	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✓
A14 SHMR (Guo et al., 2015)	✓	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
A15 Tengu (Vanhove et al., 2015)	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗	✗	✗	✗
A16 (Xie et al., 2015)	✓	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✓	✗
A17 (e Sá et al., 2015)	✓	✓	✓	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓
A18 D-Ocean (Zhuang et al., 2016)	✓	✓	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗

Software Reference Architectures	*Volume*			*Velocity*		*Variety*			*Variability*			*Veracity*			
	R1.1	R1.2	R1.3	R2.1	R2.2	R3.1	R3.2	R3.3	R4.1	R4.2	R4.3	R5.1	R5.2	R5.3	R5.4
A19 NIST (Grady et al., 2014)	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗	✓	✗	✓	✓	✓
A20 (Pääkkönen and Pakkala, 2015)	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓
A21 (Geerdink, 2015)	✓	✓	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Bolster	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2: Fulfillment of each requirement in the related work

SQL-like [A4,A10,A11,A18] or SPARQL [A1,A8] languages. Furthermore, those offering MapReduce or similar interfaces [A2,A3,A6,A13,A14,A15,A20] meet the predictive and prescriptive level (**R1.3**). HaoLap [A12] and SHMR [A14] are the only works where MapReduce is narrowed to descriptive queries.

3.2.2. Requirements on Velocity

Several architectures are capable of ingesting data streams (**R2.1**), either by dividing the architecture in specialized Batch and Real-time Layers [A2,A6,A7,A10,A11,A15,A20], by providing specific channels like data feeds [A4] or by solely considering streams as input type [A1,A8,A9]. Regarding processing of such data streams (**R2.2**), all architectures dealing with its ingestion can additionally perform processing, with the exception of AsterixDB [A4] and M3Data [A5], where data streams are stored prior to querying them.

3.2.3. Requirements on Variety

Variety is handled in diverse ways in the studied architectures. Concerning ingestion of raw data (**R3.1**), few proposals cannot deal with such requirement, either because they are narrowed to ingest specific data formats [A8,A16], or because specific wrappers need to be defined on the sources [A1,A19]. Concerning storage of raw data (**R3.2**), many architectures define views to merge and homogenize different formats into a common one (including those that do it at ingestion time) [A4,A5,A10,A12,A14,A15,A17]. On the other hand, the λ-architecture and some of the akin architectures [A2,A6,A7,A11] and [A20] are the only ones natively storing raw data. In schema management (**R3.3**), all those architectures that favored ingesting and storing raw data cannot deal with such requirement, as no additional mechanism is present to handle it. Oppositely, the ones defining unified views are able to manage them, likewise relational database schemas. There is an exception to the previous discussion, D-Ocean [A18], which defines a data model for unstructured data, hence favouring all requirements.

3.2.4. Requirements on Variability

Requirements on Variability are poorly covered among the reviewed works. Schema evolution is only handled by CQELS [A1], AsterixDB [A4] and D-Ocean [A18]. CQELS uses specific wrapper configuration files which via a user interface map new elements to ontology concepts. On the other hand, AsterixDB parses schemas at runtime. Finally, D-Ocean's unstructured data model embraces the addition of new features. Furthermore, only AsterixDB considers data evolution (**R4.2**) using adaptive query processing techniques. With respect to automatic inclusion of data sources (**R4.3**), CQELS has a service allowing wrappers to be plugged at runtime. Moreover, other architectures provide such feature as AsterixDB with the definition of external tables at runtime, [A19] providing a discovery channel or Tengu [A15] by means of an Enterprise Service Bus.

3.2.5. Requirements on Veracity

Few of the studied architectures satisfy requirements on Veracity. All works covering data provenance (**R5.1**) log the operations applied on derived data in order to be reproduced later. On the other hand, measurement of data quality (**R5.2**) is only found in [A19] and [A13], the former by storing such metadata as part of its Big Data lifecycle and the latter by tracking data quality rules that validate the stored data. Regarding data liveliness (**R5.3**), [A16] tracks it in order to boost reusage of results computed by other users. Alternatively, [A19] as part of its Preservation Management activity applies aging strategies, however it is limited to its data retention policy. Finally, with respect to data cleaning (**R5.4**) we see two different architectures. In [A5,A13,A17,A19] cleansing processes are triggered as part of the data integration phase (i.e. before being stored). Differently, [A10,A20] execute such processes on unprocessed raw data before serving them to the user.

3.3. Discussion

Besides new technological proposals, we devise two main families of works in the Big Data architectures landscape. On the one hand, those presented as an evolution of the λ-architecture [A7] after refining it [A2,A6,A10,A11,A15]; and, on the other hand, those positioned on the Semantic Web principles [A1,A8]. Some architectures aim to be of general-purpose, while others are tailored to specific domains, such as: multimedia data [A14], cloud manufacturing [A3], scientific testing [A15], Internet of Things [A2] or healthcare [A13].

It can be concluded from Table 2 that requirements related to Volume, Velocity and Variety are more fulfilled with respect to those related to Variability and Veracity. This is due to the fact, to some extent, that Volume, Velocity and partly Variety (i.e., **R3.1**, **R3.2**) are core functionalities in NOSQL systems, and thus all architectures adopting them benefit from that. Furthermore, such dimensions have a clear impact on the performance of the system. Most of the architectures based on the λ-architecture naturally fulfil them for such reason. On the other hand, partly Variety (i.e., **R3.3**), Variability and Veracity are dimensions that need to be addressed by respectively considering evolution and data governance as first-class citizens. However, this fact has an impact on the architecture as a whole, and not on individual components, hence causing such low fulfiment across the studied works.

4. Bolster: a Semantic Extension for the λ-Architecture

In this section, we present *Bolster*, an SRA solution for Big Data systems that deals with the 5 "Vs". Briefly, *Bolster* adopts the best out of the two families of Big Data architectures (i.e., λ-architecture and those relying on Semantic Web principles). Building on top of the λ-architecture, it ensures the fulfillment of requirements related to Volume and Velocity. However, in contrast to other approaches, it is capable of completely handling Variety, Variability and Veracity leveraging on Semantic Web technologies to represent machine-readable metadata, oppositely to the studied Semantic Web-based architectures representing data. We first present the methodology used to design the SRA. Next, we present the conceptual view of the SRA and describe its components.

4.1. The design of Bolster

Bolster has been designed following the framework for the design of empirically-grounded reference architectures (Galster and Avgeriou, 2011), which consists of a six-step process described as follows:

Step 1: decision on type of SRA. The first step consists on deciding the type of SRA to be designed, which is driven by its purpose. Using the characterization from (Angelov et al., 2012), we conclude that *Bolster* should be of type 5 (a preliminary, facilitation architecture designed to be implemented in multiple organizations). This entails that the purpose of its design is to facilitate the design of Big Data systems, in multiple organizations and performed by a research-oriented team.

Step 2: selection of design strategy. There are two strategies to design SRAs, from scratch or from existing architectures. We will design *Bolster* based on the two families of Big Data architectures identified in Section 3.

Step 3: empirical acquisition of data. In this case, we leverage on the Big Data dimensions (the five "V's") discussed in Section 2 and the requirements defined for each of them. Such requirements, together with the design strategy, will drive the design of *Bolster*.

Step 4: construction of SRA. The rationale and construction of *Bolster* is depicted in Section 4.2, where a conceptual view is presented. A functional description of its components is later presented in Section 4.3, and a functional example in Section 5.

Step 5: enabling SRA with variability. The goal of enabling an SRA with variability is to facilitate its instantiation towards different use cases. To this end, we provide the annotated SRA using a conceptual view as well as the description of components, which can be selectively instantiated. Later, in Section 6, we present methods for its instantiation.

Step 6: evaluation of the SRA. The last step of the design of an SRA is its evaluation. Here, and leveraging on the industrial projects where *Bolster* has been adopted, in Section 7.2, we present the results of its validation.

4.2. Adding semantics to the λ-architecture

The λ-architecture is the most widespread framework for scalable and fault-tolerant processing of Big Data. Its goal is to enable efficient real-time data management and analysis by being divided into three layers (Figure 1).

- The *Batch Layer* stores a copy of the master data set in raw format as data are ingested. This layer also pre-computes *Batch Views* that are provided to the *Serving Layer*.

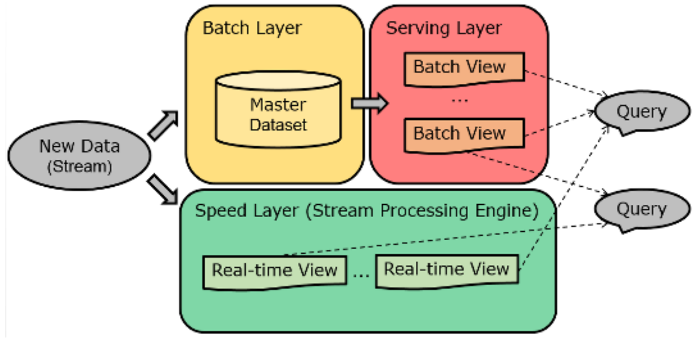

Figure 1: λ-architecture

- The *Speed Layer* ingests and processes real-time data in form of streams. Results are then stored, indexed and published in *Real-time Views*.
- The *Serving Layer*, similarly as the *Speed Layer*, also stores, indexes and publishes data resulting from the *Batch Layer* processing in *Batch Views*.

The λ-architecture succeeds at Volume requirements, as tons of heterogeneous raw data can be stored in the master data set, while fast querying through the Serving Layer. Velocity is also guaranteed thanks to the Speed Layer, since real-time views complement query results with real-time data. For these reasons, the λ-architecture was chosen as departing point for *Bolster*. Nevertheless, we identify two main drawbacks. First, as pointed out previously, it completely overlooks Variety, Variability and Veracity. Second, it suffers from a vague definition, hindering its instantiation. For example, the Batch Layer is a complex subsystem that needs to deal with data ingestion, storage and processing. However, as the λ-architecture does not define any further component of this layer, its instantiation still remains challenging. *Bolster* (Figure 2) addresses the two drawbacks identified in the λ-architecture:

- Variety, Variability and Veracity are considered first-class citizens. With this purpose, *Bolster* includes the Semantic Layer where the Metadata Repository stores machine-readable semantic annotations, in an analogous purpose as of the relational DBMS catalog.
- Inspired by the functional architecture of relational DBMSs, we refine the λ-architecture to facilitate its instantiation. These changes boil down to a precise definition of the components and their interconnections. We therefore introduce possible instantiations for each component by means of off-the-shell software or service.

Finally, note that this SRA aims to broadly cover different Big Data use cases, however it can be tailored by enabling or disabling components according to each particular context. In the following subsections we describe each layer present in *Bolster* as well as their interconnections. In bold, we highlight the

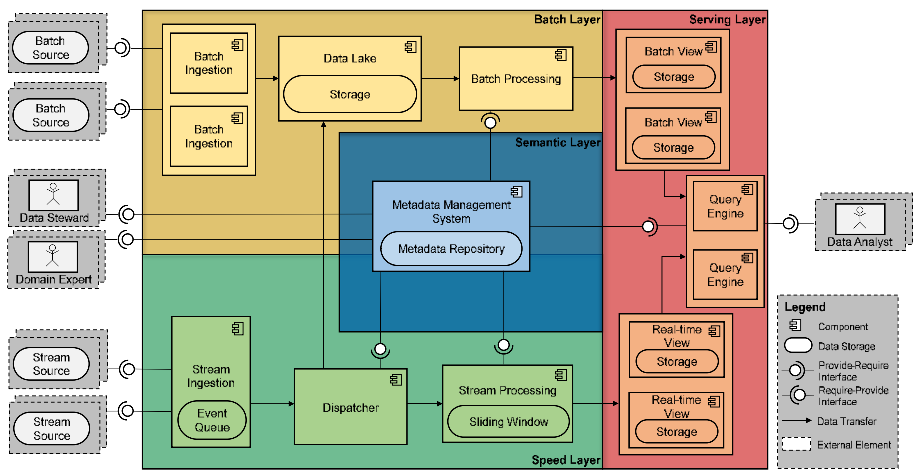

Figure 2: *Bolster* SRA conceptual view

necessary functionalities they need to implement to cope with the respective requirements.

4.3. Bolster Components

In this subsection, we present, for each layer composing *Bolster*, the list of its components and functional description.

4.3.1. Semantic Layer

The Semantic Layer (depicted blue in Figure 2) contains the Metadata Management System (MDM), the cornerstone for a semantic-aware Big Data system. It is responsible of providing the other components with the necessary information to describe and model raw data, as well as keeping the footprint about data usage. With this purpose, the MDM contains all the metadata artifacts, represented by means of RDF ontologies leveraging the benefits provided by Semantic Web technologies, needed to deal with data governance and assist data exploitation. We list below the main artifacts and refer the interested reader to (Varga et al., 2014; Bilalli et al., 2016) for further details:

1. Data analysts should work using their day-by-day vocabulary. With this purpose, the **Domain Vocabulary** contains the business concepts (e.g., `customer`, `order`, `lineitem`) and their relationships (**R5.1**).

2. In order to free data analysts from data management tasks and decouple this role from the data steward, each vocabulary term must be mapped to the system views. Thus, the MDM must be aware of the **View Schemata** (**R3.3**) and the mappings between the vocabulary and such schemata.

3. Data analysts tend to repeat the same data preparation steps prior to conducting their analysis. To enable reusability and a collaborative exploitation of the data, on the one hand, the MDM must store **Pre-processing Domain Knowledge** about data preparation rules (e.g., data cleaning, discretization, etc.) related to a certain domain (**R5.4**), and on the other hand descriptive statistics to assess data evolution (**R4.2**).

4. To deal with automatic inclusion of new data sources (**R4.3**), each ingested element must be annotated with its schema information (**R4.1**). To this end, the **Data Source Register** tracks all input data sources together with the required information to parse them, the physical schema, and each schema element has to be linked to the attributes it populates, the logical schema (**R3.3**). Furthermore, for data provenance (**R5.1**), the **Data Transformations Log** has to keep track of the performed transformation steps to produce the views, the last processing step within the Big Data system.

Populating these artifacts is a challenge. Some of them can be automatically populated and some others must be manually annotated. Nonetheless, all of these artifacts are essential to enable a centralized master metadata management and hence, fulfil the requirements related to Variety, Variability and Veracity. Analogously to database systems, data stewards are responsible of populating and maintaining such artifacts. That is why we claim for the need that the MDM provides a user friendly interface to aid such processes. Finally, note that most of the present architectural components must be able to interact with the MDM, hence it is essential that it provides language-agnostic interfaces. Moreover, such interfaces cannot pose performance bottlenecks, as doing so would highly impact in the overall performance of the system.

4.3.2. Batch Layer

This layer (depicted yellow in Figure 2) is in charge of storing and processing massive volumes of data. In short, we first encounter Batch Ingestion, responsible for periodically ingesting data from the batch sources, then the Data Lake, capable of managing large amounts of data. The last step is the Batch Processing component, which prepares, transforms and runs iterative algorithms over the data stored in the Data Lake to shape them accordingly to the analytical needs of the use-case at hand.

Batch Ingestion. Batch sources are commonly big static raw data sets that require periodic synchronizations (**R3.1**). Examples of batch sources can be relational databases, structured files, etc. For this reason, we advocate for a multiple component instantiation, as required by the number of sources and type. These components need to know which data have already been moved to the Data Lake by means of **Incremental Bulks Scheduling and Orchestration**. The MDM then comes into play as it traces this information. Interaction between the ingestion components and the MDM occurs in a two-phase manner. First, they learn which data are already stored in the Data Lake, to identify the according incremental bulk can be identified. Second, the MDM is enriched with specific information regarding the recently brought data (**R5.3**). Since Big Data systems are multi-source by nature, the ingestion components must be built to guarantee its adaptability in the presence of new sources (**R4.3**).

Data Lake. This component is composed of a **Massive Storage** system (**R1.1**). Distributed file systems are naturally good candidates as they were born to hold large volumes of data in their source format (**R3.2**). One of their main drawbacks is that its read capabilities are only sequential and no complex querying is therefore feasible. Paradoxically, this turns out to be beneficial for the Batch Processing, as it exploits the power of cloud computing.

Different file formats pursuing high performance capabilities are available, focusing on different types of workload (Munir et al., 2016). They are commonly classified as horizontal, vertical and hybrid, in an analogous fashion as row-oriented and column-oriented databases, respectively.

Batch Processing. This component models and transforms the Data Lake's files into Batch Views ready for the analytical use-cases. It is responsible to schedule and execute **Batch Iterative Algorithms**, such as sorting, searching, indexing (**R1.2**) or more complex algorithms such as PageRank, Bayesian classification or genetic algorithms (**R1.3**). The processing components, must be designed to maximize reusability by creating building blocks (from the domain-knowledge metadata artifacts) that can be reused in several views. Consequently, in order to track **Batch Data Provenance**, all performed transformations must be communicated to the MDM (**R5.1**).

Batch processing is mostly represented by the MapReduce programming model. Its drawbacks appear twofold. On one hand, when processing huge amounts of batch data, several jobs may usually need to be chained so that more complex processing can be executed as a single one. On the other hand, intermediate results from Map to Reduce phases are physically stored in hard disk, completely detracting the Velocity (in terms of response time).

Massive efforts are currently put on designing new solutions to overcome the issues posed by MapReduce. For instance, by natively including other more atomic relational algebra operations, connected by means of a directed acyclic graph; or by keeping intermediate results in main memory.

4.3.3. Speed Layer

The Speed Layer (depicted green in Figure 2) deals primarily with Velocity. Its input are continuous, unbounded streams of data with high timeliness and therefore require novel techniques to accommodate such arrival rate. Once ingested, data streams can be dispatched either to the Data Lake, in order to run historical queries or iterative algorithms, or to the Stream Processing engine, in charge of performing one-pass algorithms for real-time analysis.

Stream Ingestion. The Stream Ingestion component acts as a message queue for raw data streams that are pushed from the data sources (**R3.1**). Multiple sources can continuously push data streams (e.g., sensor or social network data), therefore such component must be able to cope with high throughput rates and scale according to the number of sources (**R2.1**). One of the key responsibilities is to enable the ingestion of all incoming data (i.e., adopt a **No Event Loss** policy). To this end, it relies on a distributed memory or disk-based storage buffer (i.e. event queue), where streams are temporarily stored.

This component does not require any knowledge about the data or schema of incoming data streams, however, for each event, it must know its source and type, for further matching with the MDM. To assure fault-tolerance and durability of results in such a distributed environment, techniques such as write-ahead logging or the two-phase commit protocol are used, nevertheless that has a clear impact on the availability of data to next components.

Dispatcher. The responsibilities of the Dispatcher are twofold. On the one hand, to ensure data quality, via MDM communication, it must register and validate that all ingested events follow the specified schema and rules for the event on hand (i.e., **Schema Typechecking** (**R4.1**, **R5.2**)). Error handling mechanisms must be triggered when an event is detected as invalid, and various mitigation plans can be applied. The simplest alternative is event rejection, however most conservative approaches like routing invalid events to the Data Lake for future reprocess can contribute to data integrity.

On the other hand, the second responsibility of the Dispatcher is to perform **Event Routing**, either to be processed in a real-time manner (i.e., to the Stream Processing component), or in a batch manner (i.e., to the Data Lake) for delayed process. In contrast to the λ-architecture, which duplicates all input streams to the Batch Layer, here only those that will be used by the processing components will be dispatched if required. Moreover, before dispatching such events, different routing strategies can influence the decision on where data is shipped, for instance by means of evaluating QoS cost models or analyzing the system workload, as done in (Kroß et al., 2015). Other approaches like sampling or load shedding can be used here, to ensure that either real-time processing or Data Lake ingestion are correctly performed.

Stream Processing. The Stream Processing component is responsible of performing **One-Pass Algorithms** over the stream of events. The presence of a summary is required as most of these algorithms leverage on in-memory stateful data structures (e.g., the Loosy Counting algorithm to compute heavy hitters, or HyperLogLog to compute distinct values). Such data structures can be leveraged to maintain aggregates over a sliding window for a certain period of time. Different processing strategies can be adopted, being the most popular tuple-at-a-time and micro-batch processing, the former providing low latency while the latter providing high throughput (**R2.2**). Similarly as the Batch Processing, this component must communicate to the MDM all transformations applied to populate Real-time Views in order to guarantee **Stream Data Provenance** (**R5.1**).

4.3.4. Serving Layer

The Serving Layer (depicted red in Figure 2) holds transformed data ready to be delivered to end-users (i.e. it acts as a set of database engines). Precisely, it is composed by Batch and Real-time Views repositories. Different alternatives exist when selecting each view engine, however as they impose a data model (e.g., relational or key-value), it is key to perform a goal-driven selection according to end-user analytical requirements (Herrero et al., 2016). It is worth noting that views can also be considered new sources, in case it is required to perform transformations among multiple data models, resembling a feedback loop. Further, the repository of Query Engines is the entry point for data analysts to achieve their analytical task, querying the views and the Semantic Layer.

Batch Views. As in the λ-architecture, we seek **Scalable and Fault-Tolerant Databases** capable to provide **Random Reads**, achieved by indexing, and the execution of **Aggregations and UDFs** (user defined functions) over large stable data sets (**R1.1**). The λ-architecture advocates for recomputing Batch Views every time a new version is available, however we claim incremental approaches should be adopted to avoid unnecessary writes and reduce processing latency. A common example of Batch View is a DW, commonly implemented in relational or columnar engines. However databases implementing other data models such as graph, key-value or documents also can serve the purpose of Batch Views. Each view must provide a high-level query language, serving as interface with the Query Engine (e.g., SQL), or a specific wrapper on top of it providing such funcionalities.

Real-time Views. As opposite to Batch Views, Real-time Views need to provide **Low Latency Querying** over dynamic and continuously changing data sets (**R2.1**). In order to achieve so, in-memory databases are currently the most suitable option, as they dismiss the high cost it entails to retrieve data from disk. Additionally, Real-Time views should support low cost of updating in order to maintain **Sketches and Sliding Windows**. Finally, similarly to Batch Views, Real-time Views must provide mechanisms to be queried, considering as well **Continuous Query Languages**.

Query Engines. Query Engines, play a crucial role to enable efficiently querying the views in a friendly manner for the analytical task on hand. Data analysts query the system using the vocabulary terms and apply domain-knowledge rules on them (**R1.2, R1.3**). Thanks to the MDM artifacts, the system must internally perform the translation from **Business Requirements to Database Queries** over Batch and Real-time Views (**R3.3**), hence making data management tasks transparent to the end-user. Furthermore, the Query Engine must provide to the user the ability for **Metadata Query and Exploration** on what is stored in the MDM (**R5.1, R5.2, R5.3**).

4.3.5. Summary

Table 3 summarizes for each component the fulfilled requirements discussed in Section 2.

Component	*Volume*			*Velocity*		*Variety*			*Variability*			*Veracity*			
	R1.1	R1.2	R1.3	R2.1	R2.2	R3.1	R3.2	R3.3	R4.1	R4.2	R4.3	R5.1	R5.2	R5.3	R5.4
Metadata Management System								✓	✓	✓	✓	✓			✓
Batch Ingestion						✓					✓			✓	
Data Lake	✓						✓								
Batch Processing		✓	✓									✓			
Stream Ingestion				✓		✓									
Dispatcher									✓				✓		
Stream Processing					✓							✓			
Batch Views	✓														
Real-time Views				✓											
Query Engines		✓	✓					✓				✓	✓	✓	

Table 3: *Bolster* components and requirements fulfilled

5. Exemplar Use Case

The goal of this section is to provide an exemplar use case to illustrate how *Bolster* would accommodate a Big Data management and analytics scenario. Precisely, we consider the online social network benchmark described in (Zhang et al., 2015). Such benchmark aims to provide insights on the stream of data provided by Twitter's Streaming API, and is characterized by workloads in media, text, graph, activity and user analytics.

5.1. Semantic representation

Figure 3 depicts a high level excerpt of the content stored in the MDM. In dark and light blue, the domain knowledge and business vocabulary respectively which has been provided by the Domain Expert. In addition, the data steward has, possibly in a semi-automatic manner (Nadal et al., 2017), registered a new source (Twitter Stream API[4]) and provided mappings for all JSON fields to the logical attributes (in red). For the sake of brevity, only the relevant subgraph of the ontology is shown. Importantly, to meet the Linked Open Data principles, this ontology should be further linked to other ontologies (e.g., the Open Provenance Model (Moreau et al., 2011)).

5.2. Data ingestion

As raw JSON events are pushed to the Stream Ingestion component, they are temporary stored in the Event Queue. Once replicated, to guarantee durability and fault tolerance, they are made available to the Dispatcher, which is aware on how to retrieve and parse them by querying the MDM. Twitter's documentation[5] warns developers that events with missing counts rarely happen. To guarantee data quality such aspect must be checked. If an invalid event is detected, it

[4] https://dev.twitter.com/streaming/overview

[5] https://dev.twitter.com/streaming/overview/processing

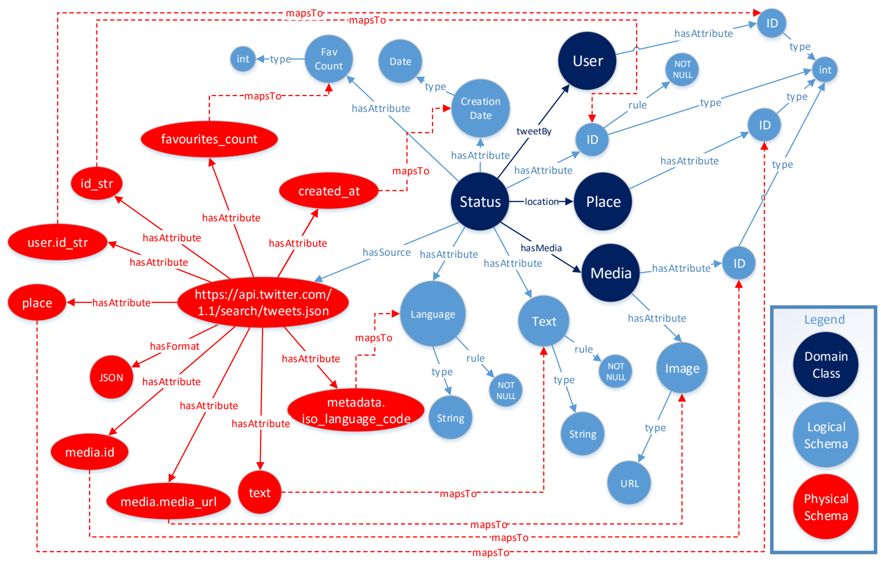

Figure 3: Excerpt of the content in the Metadata Repository

should be discarded. After this validation, the event at hand must be registered in the MDM to guarantee lineage analysis. Furthermore the Dispatcher sends the raw JSON event to the Stream Processing and Data Lake components. At this point, there is a last ingestion step missing before processing data. The first workload presented in the benchmark concerns media analytics, however as depicted in Figure 3, the API only provides the URL of the image. Hence, it is necessary to schedule a batch process periodically fetching such remote images and loading them into the Data Lake.

5.3. Data processing and analysis

Once all data are available to be processed in both Speed and Batch Layers, we can start executing the required workloads. Many of such workloads concern predictive analysis (e.g., topic modeling, sentiment analysis, location prediction or collaborative filtering). Hence, the proposed approach is to periodically refresh statistical models in an offline manner (i.e., in the Batch Layer), in order to assess predictions in an online manner (i.e., in the Speed Layer). We distinguish between those algorithms generating metadata (e.g., Latent Dirichlet Allocation (LDA)) and those generating data (e.g., PageRank). The former will store its results in the MDM using a comprehensive vocabulary (e.g., OntoDM (Panov et al., 2008)); and the latter will store them into Batch Views. Once events have been dispatched, the required statistical model has to be retrieved from the MDM to assess predictions and store outcomes into Real-time Views. Finally, as described in (Zhang et al., 2015), the prototype application provides insights based on tweets related to companies in the S&P 100 index. Leveraging on the MDM, the Query Engine is capable of generating queries to Batch and Real-time Views.

6. *Bolster* Instantiation

In this section we list a set of candidate tools, with special focus on the Apache Hadoop and Amazon Web Services ecosystems, to instantiate each component in *Bolster*. In the case when few tools from such ecosystems were available, we propose commercial tools which were considered in the industrial projects where *Bolster* was instantiated. Further, we present a method to instantiate the reference architecture. We propose a systematic scoring process driven by quality characteristics, yielding, for each component, the most suitable tool.

6.1. Available tools

6.1.1. Semantic Layer

Metadata Management System. Two different off-the-shelf open source products can instantiate this layer, namely *Apache Stanbol*[6] and *Apache Atlas*[7]. Nevertheless, the features of the former fall short for the proposed requirements of the MDM. Not surprisingly, this is due to the novel nature of *Bolster*'s Semantic Layer. *Apache Atlas* satisfies the required functionalities more naturally and it might appear as a better choice, however it is currently under heavy development as an *Apache Incubator* project. Commercial tools such as *Cloudera Navigator*[8] or *Palantir*[9] are also candidate tools.

Metadata Storage. We advocate for the adoption of Semantic Web storage technologies (i.e. triplestores), to store all the metadata artifacts. Even though such tools allow storing and reasoning over large and complex ontologies, that is not the pursued purpose here, as our aim is to allow a simple and flexible representation of machine-readable schemas. That is why triplestores serve better the purpose of such storage. *Virtuoso*[10] is at the moment the most mature triplestore platform, however other options are available such as *4store*[11] or *GraphDB*[12]. Nonetheless, given the graph nature of triples, any graph database can as well serve the purpose of metadata storage (e.g., *AllegroGraph*[13] or *Neo4j*[14]).

[6] https://stanbol.apache.org
[7] http://atlas.incubator.apache.org
[8] https://www.cloudera.com/products/cloudera-navigator.html
[9] https://www.palantir.com
[10] http://virtuoso.openlinksw.com
[11] http://4store.org
[12] http://graphdb.ontotext.com/graphdb
[13] http://allegrograph.com
[14] http://neo4j.com

6.1.2. Batch Layer

Batch Ingestion. This components highly depends on the format of the data sources, hence it is complex to derive a universal driver due to technological heterogeneity. Instantiating this component usually means developing *ad-hoc* scripting solutions adapting to the data sources as well as enabling communication with the MDM. Massive data transfer protocols such as FTP or Hadoop's *copyFromLocal*[15] will complement such scripts. However, some drivers for specific protocols exist such as *Apache Sqoop*[16], the most widespread solution to load data from/to relational sources through JDBC drivers.

Data Lake. *Hadoop Distributed File System* and *Amazon S3*[17] perfectly fit in this category, as they are essentially file systems storing plain files. Regarding data file formats, some current popular options are *Apache Avro*[18], *Yahoo Zebra*[19] or *Apache Parquet*[20] for horizontal, vertical and hybrid fragmentation respectively.

Batch Processing. *Apache MapReduce*[21] and *Amazon Elastic MapReduce*[22] are nowadays the most popular solutions. Alternatively, *Apache Spark*[23] and *Apache Flink*[24] are gaining great popularity as next generation replacement for the MapReduce model. However, to the best of our knowledge, only *Quarry* (Jovanovic et al., 2015) is capable to interact with the MDM and, based on the information there stored, automatically produce batch processes based on user-defined information requirements.

6.1.3. Speed Layer

Stream Ingestion. All tools in the family of "message queues" are candidates to serve as component for Stream Ingestion. Originated with the purpose of serving as middleware to support enterprise messaging across heterogeneous systems, they have been enhanced with scalability mechanisms to handle high ingestion rates preserving durability of data. Some examples of such systems are *Apache ActiveMQ*[25] or *RabbitMQ*[26]. However, some other tools were born following similar principles but aiming Big Data systems since its inception, being *Apache Kafka*[27] and *AWS Kinesis Firehose*[28] the most popular options.

[15] https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-common/FileSystemShell.html#copyFromLocal
[16] http://sqoop.apache.org
[17] https://aws.amazon.com/s3
[18] https://avro.apache.org
[19] http://pig.apache.org/docs/r0.9.1/zebra_overview.html
[20] https://parquet.apache.org
[21] https://hadoop.apache.org
[22] https://aws.amazon.com/elasticmapreduce
[23] http://spark.apache.org
[24] https://flink.apache.org
[25] http://activemq.apache.org
[26] https://www.rabbitmq.com
[27] http://kafka.apache.org
[28] https://aws.amazon.com/kinesis/firehose

Dispatcher. Here we look for tools that allow developers to define data pipelines routing data streams to multiple and heterogeneous destinations. It should also allow the developer to programmatically communicate with the MDM for quality checks. *Apache Flume*[29] and *Amazon Kinesis Streams*[30] are nowadays the most prevalent solutions.

Stream Processing. In contrast to Batch Processing, it is unfeasible to adopt classical MapReduce solutions considering the performance impact they yield. Thus, in-memory distributed stream processing solutions like *Apache Spark Streaming*[31], *Apache Flink Streaming*[32] and *Amazon Kinesis Analytics*[33] are the most common alternatives.

6.1.4. Serving Layer

Batch Views. A vast range of solutions are available to hold specialized views. We distinguish among three families of databases: (distributed) relational, NOSQL and NewSQL. The former is mostly represented by major vendors who evolved their traditional centralized databases into distributed ones seeking to improve its storage and performance capabilities. Some common solutions are *Oracle*[34], *Postgres-XL*[35] or *MySQL Cluster*[36]. Secondly, in the NOSQL category we might drill-down to the specific data model implemented: *Apache HBase*[37] or *Apache Cassandra*[38] for column-family key-value; *Amazon DynamoDB*[39] or *Voldemort*[40] for key-value; *Amazon Redshift*[41] or *Apache Kudu*[42] for column oriented; *Neo4j*[43] or *OrientDB*[44] for graph; and *MongoDB*[45] or *RethinkDB*[46] for document. Finally, NewSQL are high-availability main memory databases which usually are deployed in specialized hardware, where we encounter *SAP Hana*[47], *NuoDB*[48] or *VoltDB*[49].

[29] https://flume.apache.org
[30] https://aws.amazon.com/kinesis/streams
[31] http://spark.apache.org/streaming
[32] https://flink.apache.org
[33] https://aws.amazon.com/kinesis/analytics
[34] https://www.oracle.com/database
[35] http://www.postgres-xl.org
[36] https://www.mysql.com/products/cluster
[37] https://hbase.apache.org
[38] http://cassandra.apache.org
[39] https://aws.amazon.com/dynamodb
[40] http://www.project-voldemort.com/voldemort
[41] https://aws.amazon.com/redshift
[42] http://getkudu.io
[43] http://neo4j.com
[44] http://orientdb.com/orientdb
[45] https://www.mongodb.org
[46] https://www.rethinkdb.com
[47] https://hana.sap.com
[48] http://www.nuodb.com
[49] https://voltdb.com

Real-time Views. In-memory databases are currently the most popular options, for instance *Redis*[50], *Elastic*[51], *Amazon ElastiCache*[52]. Alternatively, *PipelineDB*[53] offers mechanism to query a data stream via continuous query languages.

Query Engine. There is a vast variety of tools available for query engines. OLAP engines such as *Apache Kylin*[54] provide multidimensional analysis capabilities, on the other hand solutions like *Kibana*[55] or *Tableau*[56] enable the user to easily define complex charts over the data views.

6.2. Component selection

Selecting components to instantiate *Bolster* is a typical (C)OTS (commercial off-the-shelf) selection problem (Kontio, 1996). Considering a big part of the landscape of available Big Data tools is open source or well-documented, we follow a quality model approach for their selection, as done in (Behkamal et al., 2009). To this end, we adopt the ISO/IEC 25000 SQuaRE standard (*Software Product Quality Requirements and Evaluation*) (ISO, 2011) as reference quality model. Such model is divided into characteristics and subcharacteristics, where the latter allows the definition of metrics (see ISO 25020). In the context of (C)OTS, the two former map to the hierarchical criteria set, while the latter to evaluation attributes. Nevertheless, the aim of this paper is not to provide exhaustive guidelines on its usage whatsoever, but to supply a blueprint to be tailored to each organization. Figure 4 depicts the subset of characteristics considered relevant for such selection. Note that not all subcharacteristics are applicable, given that we are assessing the selection of off-the-shelf software for each component.

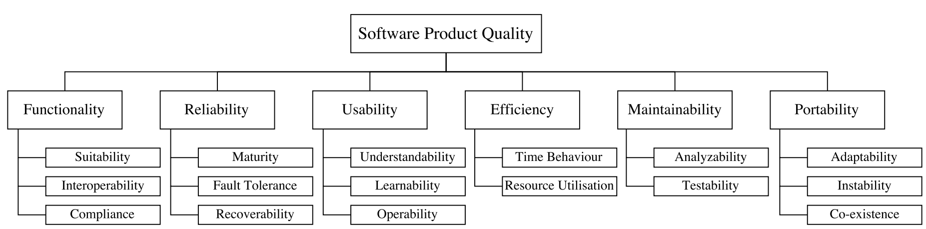

Figure 4: Selected characteristics and subcharacteristics from SQuaRE

[50] http://redis.io
[51] https://www.elastic.co
[52] https://aws.amazon.com/elasticache
[53] https://www.pipelinedb.com
[54] http://kylin.apache.org
[55] https://www.elastic.co/products/kibana
[56] http://www.tableau.com

6.2.1. *Evaluation attributes*

Previously, we discussed that ISO 25020 proposes candidate metrics for each present subcharacteristic. However, we believe that they do not cover the singularities required for selecting open source Big Data tools. Thus, in the following subsections we present a candidate set of evaluation attributes which were used in the use case applications described in Section 7. Each has associated a set of ordered values from worst to better and its semantics.

Functionality. After analyzing the artifacts derived from the requirement elicitation process, a set of target functional areas should be devised. For instance, in an agile methodology, it is possible to derive such areas by clustering user stories. Some examples of functional areas related to Big Data are: *Data and Process Mining*, *Metadata Management*, *Reporting*, *BI 2.0* or *Real-time Analysis*. *Suitability* specifically looks at such functional areas, while with the other evaluation attributes we evaluate information exchange and security concerns.

Suitability
Number of functional areas targeted in the project which benefit from its adoption.
Interoperability
1, no input/output connectors with other considered tools
2, input/output connectors available with some other considered tools
3, input/output connectors available with many other considered tools
Compliance
1, might rise security or privacy issues
2, does not raise security or privacy issues

Reliability. It deals with trustworthiness and robustness factors. *Maturity* is directly linked to the stability of the software at hand. To that end, we evaluate it by means of the Semantic Versioning Specification[57]. The other two factors, *Fault Tolerance* and *Recoverability*, are key Big Data requirements to ensure the overall integrity of the system. We acknowledge it is impossible to develop a fault tolerant system, thus our goal here is to evaluate how the system reacts in the presence of faults.

[57] http://semver.org

Maturity
1, major version zero (0.y.z)
2, public release (1.0.0)
3, major version (x.y.z)

Fault Tolerance
1, the system will crash if there is a fault
2, the system can continue working if there is a fault but data might be lost
3, the system can continue working and guarantees no data loss

Recoverability
1, requires manual attention after a fault
2, automatic recovery after fault

Usability. In this subcharacteristic, we look at productive factors regarding the development and maintenance of the system. In *Understandability*, we evaluate the complexity of the system's building blocks (e.g., parallel data processing engines require knowledge of functional programming). On the other hand, *Learnability* measures the learning effort for the team to start developing the required functionalities. Finally, in *Operability*, we are concerned with the maintenance effort and technical complexity of the system.

Understandability
1, high complexity
2, medium complexity
3, low complexity

Learnability
1, the operating team has no knowledge of the tool
2, the operating team has small knowledge of the tool and the learning curve is known to be long
3, the operating team has small knowledge of the tool and the learning curve is known to be short
4, the operating team has high knowledge of the tool

Operability
1, operation control must be done using command-line
2, offers a GUI for operation control

Efficiency. Here we evaluate efficiency aspects. *Time Behaviour* measures the performance at processing capabilities, measured by the way the evaluated tool shares intermediate results, which has a direct impact on the response time. On the other hand, *Resource Utilisation* measures the hardware needs for the system at hand, as it might affect other coexisting software.

Time Behaviour
1, shares intermediate results over the network
2, shares intermediate results on disk
3, shares intermediate results in memory

Resource Utilisation
1, high amount of resources required (on both master and slaves)
2, high amount of resources required (either on master or slaves)
3, low amount of resources required

Maintainability. It concerns continuous control of software evolution. If a tool provides fully detailed and transparent documentation, it will allow developers to build robust and fault-tolerant software on top of them (*Analyzability*). Furthermore, if such developments can be tested automatically (by means of unit tests) the overall quality of the system will be increased (*Testability*).

Analyzability
1, online up to date documentation
2, online up to date documentation with examples
3, online up to date documentation with examples and books available

Testability
1, doesn't provide means for testing
2, provides means for unit testing
3, provides means for integration testing

Portability. Finally, here we evaluate the adjustment of the tool to different environments. In *Adaptability*, we analyse the programming languages offered by the tool. *Instability* and *Co-existence* evaluate the effort required to install such tool and coexistence constraints respectively.

Adaptability
1, available in one programming language
2, available in many programming languages
3, available in different programming languages and offering API access

Instability
1, requires manual build
2, self-installing package
3, shipped as part of a platform distribution

Co-existence
1, cannot coexist with other selected tools
2, can coexist with all selected tools

6.3. Tool evaluation

The purpose of the evaluation process is, for each of the candidate tools to instantiate *Bolster*, to derive a ranking of the most suitable one according to the evaluation attributes previously described. The proposed method is based on the weighted sum model (WSM), which allows weighting criteria (w_i) in order to prioritize the different subcharacteristics. Weights should be assigned according to the needs of the organization. Table 4 depicts an example selection for the *Batch Processing* component for the use case described in Section 7.1.2. For each studied tool, the *Atomic* and *Weighted* columns indicate its unweighted (f_i) and weighted score ($w_i f_i$), respectively using a range from one to five. For each characteristic, the weighted average of each component is shown in light grey (i.e., the average of each weighted subcharacteristic $\sum_i f_i / \sum_i w_i$). Finally, in black, the final score per tool is depicted. From the exemplar case of Table 4, we can conclude that, for the posed weights and evaluated scores, *Apache Spark* should be the selected tool, in from of *Apache MapReduce* and *Apache Flink* respectively.

			Evaluated Software					
			Apache Spark		Apache MapReduce		Apache Flink	
Characteristic	Subcharacteristic	Weight	Atomic	Weighted	Atomic	Weighted	Atomic	Weighted
Functionality	Suitability	2	3	6	2	4	3	6
	Interoperability	3	3	9	1	1	1	3
	Compliance	1	2	2	2	2	2	2
			2.83		1.50		1.83	
Reliability	Maturity	1	3	3	3	3	1	1
	Fault Tolerance	5	3	15	3	15	3	15
	Recoverability	2	2	4	2	4	2	4
			2.75		2.75		2.50	
Usability	Understandability	5	2	10	3	15	2	10
	Learnability	3	4	12	4	12	2	6
	Operability	2	2	4	1	2	2	4
			2.60		2.90		2.00	
Efficiency	Time Behaviour	3	3	9	1	3	3	9
	Resource Utilisation	4	1	4	2	8	1	4
			1.86		1.57		1.86	
Maintainability	Analyzability	4	3	12	3	12	2	8
	Testability	2	2	4	1	2	1	2
			2.67		2.33		1.67	
Portability	Adaptability	3	2	6	1	3	2	6
	Instability	4	3	12	3	12	2	8
	Co-existence	1	2	2	2	2	2	2
			2.50		2.13		2.00	
			2.53		2.27		2.00	

Table 4: Example tool selection for *Batch Processing*

7. Industrial Experiences

In this section we depict three industrial projects, involving five organizations, where *Bolster* has been successfully adopted. For each project, we describe the use case context and the specific Bolster instantiation in graphical form. Finally we present the results of a preliminary validation that measure the perception of *Bolster* from the relevant industrial stakeholders.

7.1. *Use cases and instantiation*

7.1.1. *BDAL: Big Data Analytics Lab*

This project takes place in a multinational company in Barcelona[58]. It runs a data-driven business model and decision making relies on predictive models. Three main design issues were identified: (a) each department used its own processes to create data matrices, which were then processed to build predictive models. For reusability, data sets were preprocessed in ad-hoc repositories (e.g., Excel sheets), generating a data governance problem; (b) data analysts systematically performed data management tasks, such as parsing continuous variable discretization or handling missing values, with a negative impact on their efficiency; (c) data matrices computation resulted in an extremely time consuming process due to their large volumes. Thus, their update rate was usually in the range of weeks to months.

The main goal was to develop a software solution to reduce the exposure of data analysts to data management and governance tasks, as well as boost performance in data processing.

Bolster instantiation. *Bolster*'s Semantic Layer allowed the organization to overcome the data governance problem, consider additional data sources, and provide automation of data management processes. Additionally, there was a boost of performance in data processing thanks to the distributed computing and parallelism in the storage and processing of the Batch and Serving Layers. The nature of the data sources and analytical requirements did not justify the components in the Speed Layer, thus *Bolster*'s instantiation was narrowed to Batch, Semantic and Serving Layers. Figure 5 depicts the tools that compose *Bolster*'s instantiation instantiation for this use case.

7.1.2. *H2020 SUPERSEDE Project*

The SUPERSEDE[59] project proposes a feedback-driven approach for software life-cycle management. It considers user feedback and runtime data as an integral part of the design, development, and maintenance of software services and applications. The ultimate goal is to improve the quality perceived by software end-users as well as support developers and engineers to make the right software adaptation and evolution decisions. Three use cases proposed by industrial partners, namely: *Siemens AG Oesterreich* (Austria), *Atos* (Spain) and *SEnerCon GmbH* (Germany), are representative of different data-intensive application domains in the areas of energy consumption management in home automation and entertainment event webcasting.

SUPERSEDE's Big Data architecture is the heart of the analysis stage that takes place in the context of a monitor-analyze-plan-execute (MAPE) process (Kephart et al., 2007). Precisely, some of its responsibilities are (i) collecting and analyzing user feedback from a variety of sources, (ii) supporting decision

[58] No details about the company can be revealed due to non-disclosure agreements.

[59] `https://www.supersede.eu/`

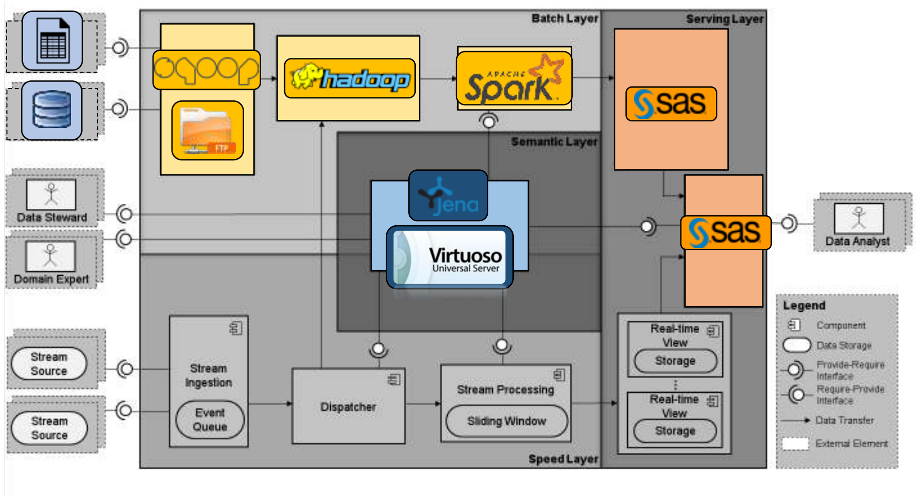

Figure 5: *Bolster* instantiation for the BDAL use case

making for software evolution and adaptation based on the collected data, and (iii) enacting the decision and assessing its impact. This set of requirements yielded the following challenges: (a) ingest multiple fast arriving data streams from monitored data and process them in real-time, for instance with sliding window operations; (b) store and integrate user feedback information from multiple and different sources; (c) use all aforementioned data in order to analyze multi-modal user feedback, identify profiles, usage patterns and identify relevant indicators for usefulness of software services. All implemented in a performance oriented manner in order to minimize overhead.

Bolster instantiation. *Bolster* allowed the definition of a data governance protocol encompassing the three use cases in a single instantiation of the architecture, while preserving data isolation. The Speed Layer enabled the ingestion of continuous data streams from a variety of sources, which were also dispatched to the Data Lake. The different analytical components in the Serving Layer allowed data analysts to perform an integrated analysis. Figure 6 depicts the tools that compose *Bolster*'s instantiation for this use case.

7.1.3. WISCC: World Information System for Chagas Control

The WISCC project funded by the World Health Organization (WHO) is part of the *Programme on Control of the Chagas disease*. The goal of this project is to control and eliminate the Chagas disease, one of the 17 diseases in the *2010 first Report on Neglected Tropical Diseases*. To this end, the aim is to build an information system serving as an integrated repository of all information, from different countries and organizations, related to the Chagas disease. Such holistic view should aid scientists to derive valuable insights and forecasts, leading to Chagas' eradication.

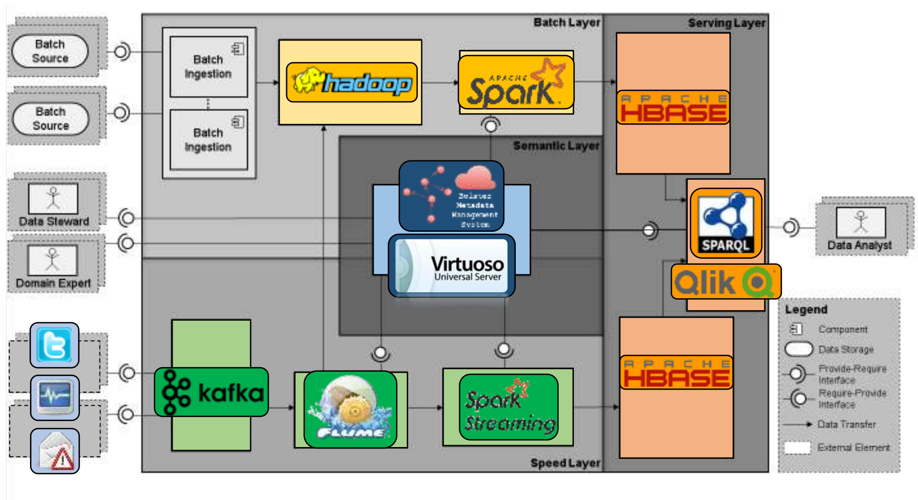

Figure 6: *Bolster* instantiation for the SUPERSEDE use case

The role of the Big Data architecture is to ingest and integrate data from a variety of data sources and formats. Currently, the big chunk of data is ingested from DHIS2[60], an information system where national ministries enter data related to inspections, diagnoses, etc. Additionally, NGOs make available similar information according to their actions. The information dealt with is continuously changing by nature at all levels: data, schema and sources. Thus, the challenge falls in the flexibility of the system to accommodate such information and the one to come. Additionally, flexible mechanisms to query such data should be defined, as future information requirements will be totally different from today's.

Bolster instantiation. Instantiating *Bolster* favored a centralized management, in the Semantic Layer, of the different data sources along with the provided schemata, a feature that facilitated the data integration and Data Lake management tasks. Similarly to the BDAL use case, the ingestion and analysis of data was performed with batch processes, hence dismissing the need to instantiate the Speed Layer. Figure 7 depicts the tools that compose *Bolster*'s instantiation for this use case.

7.1.4. Summary

In this subsection, we discuss and summarize the previously presented instantiations. We have shown how, as an SRA, *Bolster* can flexibly accomodate different use cases with different requirements by selectively instantiating its components. Due to space reasons, we cannot show the tool selection tables per

[60] https://www.dhis2.org

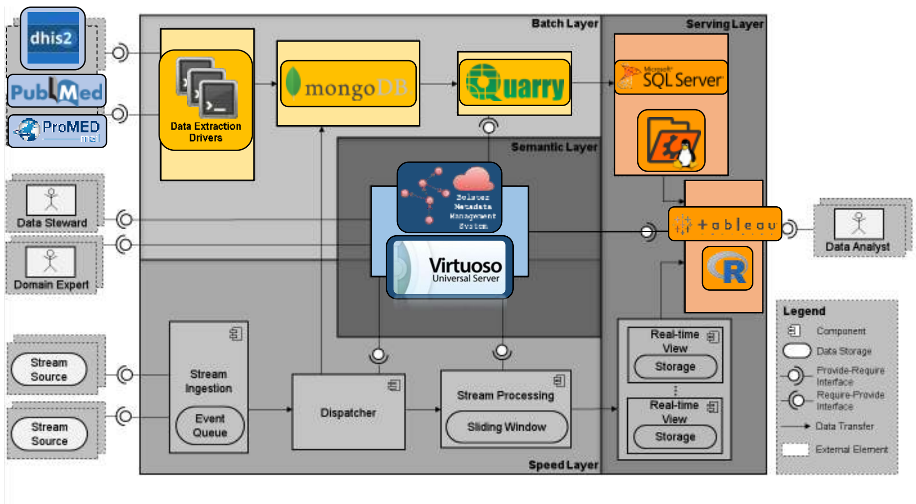

Figure 7: *Bolster* instantiation for the WISCC use case

component, instead we present the main driving forces for such selection using the dimensions devised in Section 2. Table 5 depicts the key dimensions that steered the instantiation of *Bolster* in each use case.

Use Case	*Volume*	*Velocity*	*Variety*	*Variability*	*Veracity*
BDAL	✓		✓	✓	✓
SUPERSEDE		✓	✓	✓	✓
WISCC			✓	✓	✓

Table 5: Characterization of use cases and Big Data dimensions

Most of the components have been successfully instantiated with off-the-shelf tools. However, in some cases it was necessary to develop customized solutions to satisfy specific project requirements. This was especially the case for the MDM, for which off-the-shelf tools were unsuitable in two out of three projects. It is also interesting to see that, due to the lack of connectors between components, it has been necessary to use glue code techniques (e.g., in WISCC dump files to a UNIX file system and batch loading in R). As final remark, note that the deployment of *Bolster* in all described use cases occurred in the context of research projects, which usually entail a low risk. However, in data-driven organizations such information processing architecture is the business's backbone, and adopting *Bolster* can generate risk as few components from the legacy architecture will likely be reused. This is due to the novelty in the landscape of Big Data management and analysis tools, which lead to a paradigm shift on how data are stored and processed.

7.2. Validation

The overall objective of the validation is to "assess to which extent *Bolster* leads to a perceived quality improvement in the software or service targeted in each use case". Hence, the validation of the SRA involves a quality evaluation where we investigated how Big Data practitioners perceive *Bolster*'s quality improvements. To this end, as before, we rely on SQuaRE's quality model, however now focusing on the quality-in-use model. The model is hierarchically composed by a set of characteristics and sub-characteristics. Each (sub-)characteristic is quantified by a Quality Measure (QM), which is the output of a measurement function applied to a number of Quality Measure Elements (QME).

7.2.1. Selection of participants

For each of the five aforementioned organizations, in the three use cases, a set of practitioners was selected as participants to report their perception about the quality improvements achieved with *Bolster* using the data collection method detailed in Section 7.2.2. Care was taken in selecting participants with different backgrounds (e.g., a broad range of skills, different seniority levels) and representative of the actual target population of the SRA. This is summarized in Table 6, which depicts the characteristics of the respondents in each organization. Recall that the SUPERSEDE project involves three industrial partners, hence we refer by SUP-1, SUP-2 and SUP-3 to, respectively, *Siemens*, *Atos* and *SEnerCon*.

ID	Org.	Function	Seniority	Specialties
#1	BDAL	Data analyst	Senior	Statistics
#2	BDAL	SW architect	Junior	Non-relational databases, Java
#3	SUP-1	Research scientist	Senior	Statistics, machine learning
#4	SUP-1	Key expert	Senior	Software engineering
#5	SUP-1	SW developer	Junior	Java, security
#6	SUP-1	Research scientist	Senior	Stream processing, semantic web
#7	SUP-2	Dev. team head	Senior	CDN, relational databases
#8	SUP-2	Project manager	Senior	Software engineering
#9	SUP-3	SW developer	Junior	Web technologies, statistics
#10	SUP-3	SW developer	Junior	Java, databases
#11	SUP-3	SW architect	Senior	Web technologies, project leader
#12	WISCC	SW architect	Senior	Statistics, software engineering
#13	WISCC	Research scientist	Senior	Non-relational databases, semantic web
#14	WISCC	SW developer	Junior	Java, web technologies

Table 6: List of participants per organization

7.2.2. Definition of the data collection methods

The quality characteristics were evaluated by means of questionnaires. In other words, for each characteristic (e.g., trust), the measurement method was the question whether a participant disagrees or agrees with a descriptive statement. The choice of the participant (i.e., the extent of agreement in a specific rating scale) was the QME. For each characteristic, a variable numbers of QMEs were collected (i.e., one per participant). The final QM was represented by the mean opinion score (MOS), computed by the measurement function $\sum_i^N QME_i/N$, where N is the total number of participants. We used a 7-values rating scale, ranging from 1 strongly disagree to 7 strongly agree. Table 7 depicts the set of questions in the questionnaire along with the quality subcharacteristic they map to.

Subcharacteristic	Question
Usefulness	• The presented Big Data architecture would be useful in my UC
Satisfaction	• Overall I feel satisfied with the presented architecture
Trust	• I would trust the Big Data architecture to handle my UC data
Perceived Relative Benefit	• Using the proposed Big Data architecture would be an improvement with respect to my current way of handling and analyzing UC data
Functional Completeness	• In general, the proposed Big Data architecture covers the needs of the UC (subdivided into user stories)
Functional Appropriateness	• The proposed Big Data architecture facilitates the storing and management of the UC data • The proposed Big Data architecture facilitates the analysis of historical UC data • The proposed Big Data architecture facilitates the real-time analysis of UC data stream • The proposed Big Data architecture facilitates the exploitation of the semantic annotation of UC data • The proposed Big Data architecture facilitates the visualization of UC data statistics
Functional Correctness	• The extracted metrics obtained from the Big Data architecture (test metrics) match the results rationally expected
Willingness to Adopt	• I would like to adopt the Big Data architecture in my UC

Table 7: Validation questions along with the subcharacteristics they map to

7.2.3. Execution of the validation

The heterogeneity of organizations and respondents called for a strict planning and coordination for the validation activities. A thorough time-plan was elaborated, so as to keep the progress of the evaluation among use cases. The actual collection of data spanned over a total duration of three weeks. Within these weeks, each use case evaluated the SRA in a 3-phase manner:

1. *(1 week)*: A description of Bolster in form of an excerpt of Section 4 of this paper was provided to the respondents, as well as access to the proposed

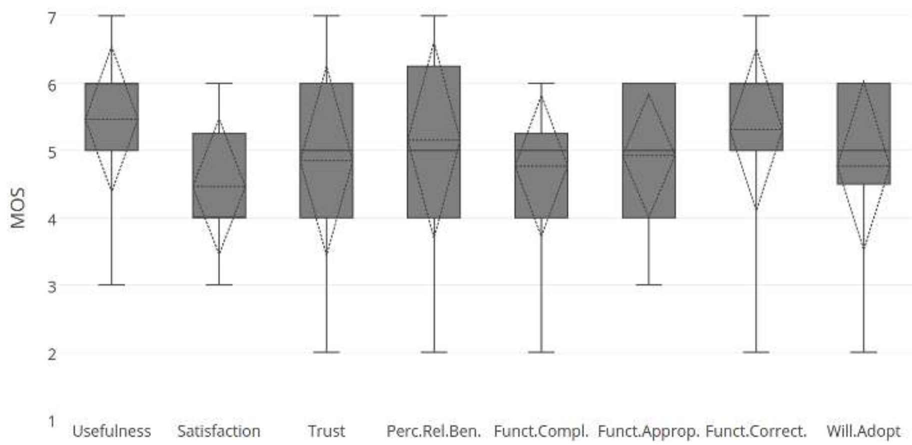

Figure 8: Validation per Quality Factor

solution tailored to each organization.

2. *(1 hour)*: For each organization, a workshop involving a presentation on the SRA and a Q&A session was carried out.

3. *(1 day)*: The questionnaire was provided to each respondent to be answered within a day after the workshop.

Once the collection of data was completed, we digitized the preferences expressed by the participants in each questionnaire. We created summary spreadsheets merging the results for its analysis.

7.2.4. Analysis of validation results

Figure 8 depicts, by means of boxplots, the aggregated MOS for all respondents (we acknowledge the impossibility to average ordinal scales, however we consider them as their results fall within the same range). The top and bottom boxes respectively denote the first and third quartile, the solid line the median and the whiskers maximum and minimum values. The dashed line denotes the average, and the diamond shape the standard deviation. Note that *Functional Appropriateness* is aggregated into the average of the 5 questions that compose it, and functional completeness is aggregated into the average of multiple user-stories (a variable number depending on the use case).

We can see that, when taking the aggregated number, none of the characteristics scored below the mean of the rating scale (1-7) indicating that *Bolster* was on average well-perceived by the use cases. Satisfaction sub-characteristics (i.e., Satisfaction, Trust, and Usefulness) present no anomaly, with usefulness standing out as the highest rated one. As far as regards Functional Appropriateness, *Bolster* was perceived to be overall effective, with some hesitation with regard to the functionality offered for the semantic exploitation of the data. All other scores are considerably satisfactory. The SRA is marked as functionally complete, and correct, and expected to bring benefits in comparison to current techniques used in the use cases. Ultimately this leads to a large intention to use.

Discussion. We can conclude that generally user's perception is positive, being most answers in the range from *Neutral* to *Strongly Agree*. The preliminary assessment shows that the potential of the Bolster SRA is recognized also in the industry domain and its application is perceived to be beneficial in improving the quality-in-use of software products. It is worth noting, however, that some respondents showed reluctancy regarding the Semantic Layer in *Bolster*. We believe this aligns with the fact that Semantic Web technologies have not yet been widely adopted in industry. Thus, lack of known successful industrial use cases may raise caution among potential adopters.

8. Conclusions

Despite their current popularity, Big Data systems engineering is still in its inception. As any other disruptive software-related technology, the consolidation of emerging results is not easy and requires the effective application of solid software engineering concepts. In this paper, we have focused on an architecture-centric perspective and have defined an SRA, *Bolster*, to harmonize the different components that lie in the core of such kind of systems. The approach uses the semantic-aware strategy as main principle to define the different components and their relationships. The benefits of *Bolster* are twofold. On the one hand, as any SRA, it facilitates the technological work of Big Data adopters by providing a unified framework which can be tailored to a specific context instead of a set of independent components that are glued together in an ad-hoc manner. On the other hand, as a semantic-aware solution, it supports non-expert Big Data adopters in the definition and exploitation of the data stored in the system by facilitating the decoupling of the data steward and analyst profiles. However, we anticipate that in the long run, with the maturity of such technologies, the role of software architect will be replaced in favor of the database administrator. In this initial deployment, *Bolster* includes components for data management and analysis as a first step towards the systematic development of the core elements of Big Data systems. Thus, *Bolster* currently maps to the role played by a relational DBMS in traditional BI systems. As future work, we foresee the need to design a generic tool providing full-fledged functionalities for Metadata Management System.

Acknowledgements

We thank Gerhard Engelbrecht for his assistance in setting up the validation process, and Silverio Martínez for his comments and insights that helped to improve this paper. This work was partly supported by the H2020 SUPERSEDE project, funded by the EU Information and Communication Technologies Programme under grant agreement no 644018, and the GENESIS project, funded by the Spanish Ministerio de Ciencia e Innovación under project TIN2016-79269-R.

9. References

Agrawal, D., Das, S., El Abbadi, A., 2011. Big Data and Cloud Computing: Current State and Future Opportunities. In: EDBT 2011.

Alsubaiee, S., Altowim, Y., Altwaijry, H., Behm, A., Borkar, V. R., Bu, Y., Carey, M. J., Cetindil, I., Cheelangi, M., Faraaz, K., Gabrielova, E., Grover, R., Heilbron, Z., Kim, Y., Li, C., Li, G., Ok, J. M., Onose, N., Pirzadeh, P., Tsotras, V. J., Vernica, R., Wen, J., Westmann, T., 2014. AsterixDB: A Scalable, Open Source BDMS. PVLDB 7 (14), 1905–1916.

Angelov, S., Grefen, P. W. P. J., Greefhorst, D., 2012. A Framework for Analysis and Design of Software Reference Architectures. Information & Software Technology 54 (4), 417–431.

Aufaure, M., 2013. What's Up in Business Intelligence? A Contextual and Knowledge-Based Perspective. In: ER 2013.

Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J., 2002. Models and Issues in Data Stream Systems. In: PODS 2002.

Batini, C., Rula, A., Scannapieco, M., Viscusi, G., 2015. From Data Quality to Big Data Quality. J. Database Manag. 26 (1), 60–82.

Behkamal, B., Kahani, M., Akbari, M. K., 2009. Customizing ISO 9126 Quality Model for Evaluation of B2B Applications. Information & Software Technology 51 (3), 599–609.

Bilalli, B., Abelló, A., Aluja-Banet, T., Wrembel, R., 2016. Towards Intelligent Data Analysis: The Metadata Challenge. In: IoTBD 2016. pp. 331–338.

Brewer, E. A., 2000. Towards Robust Distributed Systems (abstract). In: PODC 2000.

Chen, C. L. P., Zhang, C., 2014. Data-intensive Applications, Challenges, Techniques and Technologies: A Survey on Big Data. Inf. Sci. 275, 314–347.

e Sá, J. O., Martins, C., Simões, P., 2015. Big Data in Cloud: A Data Architecture. In: WorldCIST 2015.

Esteban, D., 2016. Interoperability and Standards in the European Data Economy - Report on EC Workshop. European Commission.

Fernandez, R. C., Pietzuch, P., Kreps, J., Narkhede, N., Rao, J., Koshy, J., Lin, D., Riccomini, C., Wang, G., 2015. Liquid: Unifying Nearline and Offline Big Data Integration. In: CIDR 2015.

Fox, G., Chang, W., 2015. NIST Big Data Interoperability Framework: Volume 3, Use Case and General Requirements. NIST Special Publication (1500-3).

Galster, M., Avgeriou, P., 2011. Empirically-grounded Reference Architectures: A Proposal. In: QoSA+ISARCS 2011. pp. 153–158.

Gani, A., Siddiqa, A., Shamshirband, S., Hanum, F., 2016. A Survey on Indexing Techniques for Big Data: Taxonomy and Performance Evaluation. Knowl. Inf. Syst. 46 (2), 241–284.

Garcia-Molina, H., Ullman, J. D., Widom, J., 2009. Database Systems - The Complete Book (2. ed.). Pearson Education.

García, S., Romero, O., Raventós, R., 2016. DSS from an RE Perspective: A Systematic Mapping. Journal of Systems and Software 117, 488 – 507.

Geerdink, B., 2015. A Reference Architecture for Big Data Solutions - Introducing a Model to Perform Predictive Analytics Using Big Data Technology. IJBDI 2015 2 (4), 236–249.

Giacometti, A., Marcel, P., Negre, E., 2008. A Framework for Recommending OLAP Queries. In: DOLAP 2008.

Gorton, I., Klein, J., 2015. Distribution, Data, Deployment: Software Architecture Convergence in Big Data Systems. IEEE Software 32 (3), 78–85.

Grady, N. W., Underwood, M., Roy, A., Chang, W. L., 2014. Big Data: Challenges, Practices and Technologies: NIST Big Data Public Working Group Workshop at IEEE Big Data 2014. In: IEEE Big Data 2014.

Gray, J., Liu, D. T., Nieto-Santisteban, M. A., Szalay, A. S., DeWitt, D. J., Heber, G., 2005. Scientific Data Management in the Coming Decade. SIGMOD Record 34 (4), 34–41.

Grosskurth, A., Godfrey, M. W., 2005. A reference architecture for web browsers. In: ICSM 2005. pp. 661–664.

Guo, K., Pan, W., Lu, M., Zhou, X., Ma, J., 2015. An Effective and Economical Architecture for Semantic-based Heterogeneous Multimedia Big Data Retrieval. Journal of Systems and Software 102, 207–216.

Harry, M. J., Schroeder, R. R., 2005. Six Sigma: The Breakthrough Management Strategy Revolutionizing the World's Top Corporations. Broadway Business.

Herrero, V., Abelló, A., Romero, O., 2016. NOSQL Design for Analytical Workloads: Variability Matters. In: ER 2016. pp. 50–64.

Interlandi, M., Shah, K., Tetali, S. D., Gulzar, M., Yoo, S., Kim, M., Millstein, T. D., Condie, T., 2015. Titian: Data Provenance Support in Spark. PVLDB 9 (3), 216–227.

Ionescu, B., Ionescu, D., Gadea, C., Solomon, B., Trifan, M., 2014. An Architecture and Methods for Big Data Analysis. In: SOFA 2014.

ISO, 2011. IEC25010: 2011 Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)–System and software quality models.

Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., Shahabi, C., 2014. Big Data and its Technical Challenges. Commun. ACM 57 (7), 86–94.

Jovanovic, P., Romero, O., Simitsis, A., Abelló, A., Candón, H., Nadal, S., 2015. Quarry: Digging Up the Gems of Your Data Treasury. In: EDBT 2015.

Kephart, J., Chess, D., Boutilier, C., Das, R., Kephart, J. O., Walsh, W. E., 2007. An Architectural Blueprint for Autonomic Computing.

Khatri, V., Brown, C. V., 2010. Designing Data Governance. Commun. ACM 53 (1), 148–152.

Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering.

Kontio, J., 1996. A Case Study in Applying a Systematic Method for COTS Selection. In: ICSE 1996. pp. 201–209.

Kroß, J., Brunnert, A., Prehofer, C., Runkler, T. A., Krcmar, H., 2015. Stream Processing on Demand for Lambda Architectures. In: EPEW 2015.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., Leaf, D., 2012. NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology.

Madhavji, N. H., Miranskyy, A. V., Kontogiannis, K., 2015. Big Picture of Big Data Software Engineering: With Example Research Challenges. In: BIGDSE 2015.

Martínez-Fernández, S., Ayala, C. P., Franch, X., Nakagawa, E. Y., 2015. A Survey on the Benefits and Drawbacks of AUTOSAR. In: WASA 2015.

Martínez-Prieto, M. A., Cuesta, C. E., Arias, M., Fernández, J. D., 2015. The Solid Architecture for Real-time Management of Big Semantic Data. Future Generation Comp. Syst. 47, 62–79.

Marz, N., Warren, J., 2015. Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co.

Meijer, E., Bierman, G. M., 2011. A Co-relational Model of Data for Large Shared Data Banks. Commun. ACM 54 (4), 49–58.

Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P. T., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E. G., den Bussche, J. V., 2011. The Open Provenance Model core specification (v1.1). Future Generation Comp. Syst. 27 (6), 743–756.

Munir, R. F., Romero, O., Abelló, A., Bilalli, B., Thiele, M., Lehner, W., 2016. ResilientStore: A Heuristic-Based Data Format Selector for Intermediate Results. In: MEDI 2016. pp. 42–56.

Nadal, S., Herrero, V., Romero, O., Abelló, A., Franch, X., Vansummeren, S., 2016. Details on Bolster - State of the Art. URL www.essi.upc.edu/~snadal/Bolster_SLR.html

Nadal, S., Romero, O., Abelló, A., Vassiliadis, P., Vansummeren, S., 2017. An Integration-Oriented Ontology to Govern Evolution in Big Data Ecosystems. In: DOLAP 2017.

Ordonez, C., 2010. Statistical Model Computation with UDFs. IEEE Trans. Knowl. Data Eng. 22 (12), 1752–1765.

Özsu, M. T., Valduriez, P., 2011. Principles of Distributed Database Systems, Third Edition. Springer.

Pääkkönen, P., Pakkala, D., 2015. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. Big Data Research 2 (4), 166–186.

Panov, P., Dzeroski, S., Soldatova, L. N., 2008. OntoDM: An Ontology of Data Mining. In: ICDM 2008.

Phuoc, D. L., Nguyen-Mau, H. Q., Parreira, J. X., Hauswirth, M., 2012. A Middleware Framework for Scalable Management of Linked Streams. J. Web Sem. 16, 42–51.

Qanbari, S., Zadeh, S. M., Vedaei, S., Dustdar, S., 2014. CloudMan: A Platform for Portable Cloud Manufacturing Services. In: IEEE Big Data 2014.

Russom, P., 2011. Big Data Analytics. TDWI Best Practices Report, Fourth Quarter, 6.

Sharda, R., Asamoah, D. A., Ponna, N., 2013. Business Analytics: Research and Teaching Perspectives. In: ITI 2013.

Song, J., Guo, C., Wang, Z., Zhang, Y., Yu, G., Pierson, J., 2015. HaoLap: A Hadoop Based OLAP System for Big Data. Journal of Systems and Software 102, 167–181.

Stonebraker, M., 2012. What Does ‘Big Data’ Mean. BLOG@ACM.

Stonebraker, M., 2014. Why the ’Data Lake’ is Really a ’Data Swamp’. BLOG@ACM.

Terrizzano, I., Schwarz, P. M., Roth, M., Colino, J. E., 2015. Data Wrangling: The Challenging Journey from the Wild to the Lake. In: CIDR 2015.

Tsai, C.-W., Lai, C.-F., Chao, H.-C., Vasilakos, A. V., 2015. Big Data Analytics: a Survey. Journal of Big Data 2 (1), 1–32.

Twardowski, B., Ryzko, D., 2014. Multi-agent Architecture for Real-Time Big Data Processing. In: IEEE/WIC/ACM 2014.

Vanhove, T., van Seghbroeck, G., Wauters, T., Turck, F. D., Vermeulen, B., Demeester, P., 2015. Tengu: An Experimentation Platform for Big Data Applications. In: ICDCS 2015.

Varga, J., Romero, O., Pedersen, T. B., Thomsen, C., 2014. Towards Next Generation BI Systems: The Analytical Metadata Challenge. In: DaWaK 2014.

Villari, M., Celesti, A., Fazio, M., Puliafito, A., 2014. AllJoyn Lambda: An Architecture for the Management of Smart Environments in IoT. In: SMARTCOMP 2014.

Wang, Y., Kung, L., Ting, C., Byrd, T. A., 2015. Beyond a Technical Perspective: Understanding Big Data Capabilities in Health Care. In: HICSS 2015.

Weyrich, M., Ebert, C., 2016. Reference architectures for the internet of things. IEEE Software 33 (1), 112–116.

Xie, Z., Chen, Y., Speer, J., Walters, T., Tarazaga, P. A., Kasarda, M., 2015. Towards Use And Reuse Driven Big Data Management. In: ACM/IEEE-CE 2015.

Yang, F., Merlino, G., Léauté, X., 2015. The RADStack: Open Source Lambda Architecture for Interactive Analytics.

Zeng, K., Yang, J., Wang, H., Shao, B., Wang, Z., 2013. A Distributed Graph Engine for Web Scale RDF Data. PVLDB 6 (4), 265–276.

Zhang, R., Manotas, I., Li, M., Hildebrand, D., 2015. Towards a Big Data Benchmarking and Demonstration Suite for the Online Social Network Era with Realistic Workloads and Live Data. In: BPOE 2015.

Zhuang, Y., Wang, Y., Shao, J., Chen, L., Lu, W., Sun, J., Wei, B., Wu, J., 2016. D-Ocean: An Unstructured Data Management System for Data Ocean Environment. Frontiers of Computer Science 10 (2), 353–369.