BiDArch: BigData Architect, Generator of Big Data Solution Architectures

Julio Sosa and Claudia Jiménez-Guarín(✉)

Systems and Computing Engineering Department, School of Engineering, Universidad de los Andes, Colombia Carrera 1 Este M°19A-40 Oficina ML772, 111711 Bogotá, Colombia
{jm.sosa,cjimenez}@uniandes.edu.co

Abstract. The design of highly scalable architectures based on the Hadoop ecosystem is highly complex due to the wide and dynamic portfolio of available tools. The feasible solution generator for big data problems, BiDArch, allows to automatically establish a feasible architecture from the characterization of a problem in terms of tools features. This process enriches the solution proposed from semantic knowledge allowing the architect to not require a thorough knowledge of the ecosystem, getting him to focus mainly on the characterization of his problem or design goal.

Keywords: Big Data solutions · Hadoop ecosystem · Feasible architecture generator

1 Introduction

In the scenario in which man generates heterogeneous information that can be structured or not, transmitted massively at high speed; new challenges and problems related to the discovery, storage, processing, and analysis of such information arise. A new technological and scientific boom emerges around a concept called *big data* and is interested transversely in this problematic [1, p. 3]. However, in treatable space of these problems, a set of requirements has been identified in terms of scalability, consistency and availability in information flows.

There is not only a high complexity around the problems, but also the functional and non-functional requirements impact on the elaboration of complex solution strategies that orchestrate some technology components and tools from an overwhelming set of potential candidates—with several possible configurations—to obtain a feasible solution architecture to the problem.

Hadoop represents the core of a vast tools ecosystem to build *big data* solutions. This ecosystem, which initially was limited to a base of no more than a dozen tools, today is made up of more than 60 tools [2] and *plugins* such as distributed file systems, multi-facet distributed frameworks, NoSQL repositories with different data models and specialized tools for activities like *machine learning*, deployment of distributed systems, service programming and *workflows*.

A. Solano and H. Ordoñez (Eds.): CCC 2017, CCIS 735, pp. 17–31, 2017.
DOI: 10.1007/978-3-319-66562-7_2

The existing difficulty in establishing a connection that bridges the gap between the complex domain of *big data* problems and the related knowledge to achieve an adequate tools selection to solve them, is the main motivator for developing an artifact that provides feasible solution architectures for problems using Hadoop ecosystem tools.

This problem is of great interest to any organization that intends to obtain built-in *big data* solutions, in order to provide additional value in its business processes. This problematic worries software architects who are directly faced with deciding what tools should be used and how they should be deployed, and finally, software developers who build these solutions.

The main contribution of this work is the construction of a feasible solution architecture automatic generator for problems in the domain of *big data* using Hadoop ecosystem tools. The requirements demanded by these problems are defined in terms of the functionalities offered by the tools. Another significant contribution is the development of an ontology that provides a description of a set of ecosystem tools.

2 The Complexity of Dealing with *Big Data* Problems and the Hadoop Ecosystem

Within the domain of *big data*, there are many use cases such as data exploration, business operations analysis, real-time predictive analysis, datawarehousing operations, fraud detection and risk management among many others [1, p. 1]; They focus on different sectors of industry such as financial services, health care, telecommunications and cybersecurity, among many others.

Naturally, each one of these use cases demands a high and dynamic amount of functional requirements that are aligned with the particular interests of a particular business sector, and unfortunately, are completely heterogeneous. For this reason, the complexity of performing the characterization of all problem instances achieving the functional requirements integration for all business processes is intractable. However, this whole heterogeneous set of functional requirements needs to rely on some combination of information life cycle activities in the *big data*. The above mentioned indicates that solutions will always need to discover, collect, process, analyze and visualize information flows that have enormous magnitude in dimensions of volume, speed, variety and veracity [3].

Considering that the implementation of these activities is effectively achieved with a combination of ecosystem tools, there is an immediate need to describe the tools ecosystem in terms of features that are useful to support the activities demanded directly by the functional requirements of every use case.

The problem characterization is not the only one complex stuff. The ecosystem itself also shows a high complexity, due to its constant and continuous evolution [4]. This complexity hinders the process of selecting and characterizing an initial subset of tools. This ecosystem is made up of a wide range of software projects in two licensing flavors (free and open source) [2]. Because of this

main characteristic, the elements of this ecosystem mutate with ease incubating new tools, they present wide differences in the level of maturity of the same, they produce frequent updates that impact both as an accelerated growth of the wealth of offered functionalities as in compatibility issues with other tools. In addition, the construction of specialized tools is enabled in a particular domain (scientific or commercial), such as MLlib [5] and Mahout [6] in the context of *machine learning*.

Furthermore, the identification and representation of the functionalities offered by tools is also complex, in a way that they can be abstracted to represent requirements that describe traditional problem instances.

To achieve the implementation of an effective solution, an in-depth analysis of the factors and variables is required to allow that a large variety of problems be represented and the tools selection process be measured. It is also necessary to represent explicitly the useful and relevant knowledge for designing solution architectures to these problems. These solution architectures must be built based on requirements, factors and variables considered in the characterization of a given problem and the benefits offered by the tools in the ecosystem.

BiDArch, the architecture generator presented in this work provides an important support for software architects in small and medium-sized companies that are interested in building *in-house* solutions, allowing them to characterize problem instances and providing a feasible solution architecture that is justified and enriched with a list of configuration recommendations and a knowledge related to the set of available interfaces for solution integration.

3 BiDArch: Automatic Generator of Feasible Architectures from *Big Data* Problem Specifications

The wide variety of problem instances with different processes and business interests generates a huge and complex set of requirements related to the nature of the activities carried out by each business sector that experiences these challenges. For instance, the functional requirements of an application to perform fraud detection in the financial sector do not match the requirements of a platform that wishes to integrate several data sources to provide a 360° view of a company and eliminate information silos.

However, there is a set of common requirements that are related to the features of a component or tool in a big data system and serve as input to support the requirements of each challenge in a particular business sector. Within these requirements, there are some aspects that are common to all tools such as scalability, ease of use, cost of development, cost of maintenance and feature richness, among many others. Also, there are specific requirements related to the activities that are managed by tools and contribute to orchestrate a solution to the problem or challenge of interest. In this group, there are needs around the tasks of storage, ingestion, extraction, processing and analysis of large datasets.

The purpose of BiDArch is to match the requirements of a problem instance with the features supported by the selected tools that appear in Table 1 to construct a feasible architecture with the more related tools.

Table 1. Hadoop ecosystem selected tools.

Context	Tool
Data storage	Apache Hadoop HDFS
	Apache HBase
Data ingestion	Apache Flume
	Apache Kafka
	Apache Sqoop
Data processing	Apache Hadoop MapReduce
	Apache Spark
	Apache Pig
	Apache Crunch
	Apache Cascading
	Apache Hive
	Apache Impala
Orchestration	Apache Oozie

The selection of these tools is performed considering aspects such as a high degree of maturity, stability, coverage, verifiable effectiveness in cases of success and integration with the ecosystem itself.

To achieve the goal, a phase-segmented solution strategy is planned as shown in Fig. 1.

Fig. 1. BiDarch solution strategy phases.

The phases of the proposed strategy are based on the interaction of a series of macro processes that are supported on several models. Each one of the strategy phases is detailed below.

3.1 Phase 1: Tools Description Repository Generation

After a deep study and analysis of the official documentation of the list of tools presented in Table 1, the process of generating an ontology that encapsulates the tool description repository is supported by a **tool model** designed with the aim of representing a set of features that defines properly every single tool contained into the repository.

The selection of these tools is not intended to be extensive in relation with the huge and growing number of existing free software tools around the Hadoop

ecosystem, but prioritizes the depth in the study of more relevant tools. It is important to note that all of these tools are projects of the ASF (Apache Software Foundation) community.

3.2 Phase 2: Problem Instances Specification

The main objective of this phase is to describe any problem that a BiDArch user can define in terms of the features supported by the tools. In this phase, the **problem specification process** uses the **problem model** (which is based on the **generic requirement model**) and the tools description repository generated in the previous phase, to build a problem instance.

The main elements on this phase are set out below.

Generic Requirement Model: A generic requirement is modeled as a triple composed of: a feature r_f, a value to satisfy r_v and a weight r_w that allows to establish an order relationship between the problem requirements relevance. This can be represented as follows.

$$r = (r_f, r_v, r_w) \tag{1}$$

The required feature r_f belongs to the set of tool features contained in the description repository. The domain of r_v can be **boolean** or **categorical** and it is in function of a particular *feature* r_f. On the other hand, the value of r_w satisfies the constraint $r_w \in \{1, 2, 3...10\}$ and allows to set up the relevance of a requirement with respect to others during the tools selection process.

For instance, considering for a problem p, the need to highlight *richness of functionalities* and *high scalability* features in contrast to *encryption support* and *support level* features for the selected tools; this scenario can be modeled as presented in Table 2.

Problem Model: Let P be the space of treatable problems with the Hadoop ecosystem tools, AG a set of generic attributes of the problem, RG a set of general requirements and AR a set of the required activities. A problem instance is modeled as follows:

$$P = \{p | p = (AG, RG, AR)\} \tag{2}$$

Table 2. Problem requirements example.

r_i	r_f	r_v	r_w
r_1	Scalability	High	8
r_2	Support level	Medium	3
r_3	Feature richness	High	9
r_4	Encryption support	True	4

$$AG = \{ag_1, ag_2, ag_3, ..., ag_{l-1}, ag_l\} \tag{3}$$

$$RG = \{rg_1, rg_2, rg_3, ..., rg_{m-1}, rg_m\} \tag{4}$$

$$AR = \{ar_1, ar_2, ar_3, ..., ar_{n-1}, ar_n\} \tag{5}$$

The collection of generic attributes, AG, is nothing more than a set of descriptors that contains *metadata* and relevant information of the problem. This information facilitates the problems identification and quering.

The set of general requirements, RG, contains the requirements rg_i that are common and independent of the particularity of a specific problem. These types of requirements are built on needs that are transversal to all problem instances and are finally based on features of any tool in the repository.

The problem required activities suggest special and specific requirements. These are framed within the tasks that are normally managed by ecosystem tools such as storage, movement (ingestion and extraction) and scalable processing — in its several facets— of large datasets.

Any required activity ar_i from the collection of activities AR can be represented as an ordered pair composed of an activity a_i directly related to the traditional tasks managed by the Hadoop ecosystem tools and a set of special requirements RE_i related to a particular activity a_i as shown below:

$$ar_i = (a_i, RE_i) \tag{6}$$

$$RE_i = \{re_j\} \tag{7}$$

Problem Specification Process: This process consists of a sequence of stages developed under the problem model previously raised. These stages seek to characterize the problem in terms of generic attributes, a series of general requirements, and a list of required activities that are subject to particular requirements. Figure 2 illustrates the various stages of this process.

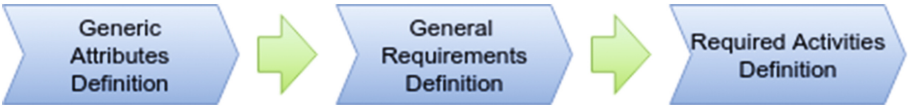

Fig. 2. Stages of the problem specification process.

3.3 Phase 3: Feasible Architectures Generation

The phase of feasible architectures generation corresponds to the final stage of the strategy and its purpose is to instantiate a feasible solution architecture diagram. To reach that goal, the previously described tool description repository and a proposed **reference architecture model** are used as inputs.

The most important components of this phase of the strategy are detailed below:

Reference Architecture Model: Any feasible architecture can be modeled as a non-empty set of tools that intercommunicate or share services through interfaces. In addition, a set of configuration recommendations are added that generate an important contribution in the solution implementation, as well as in solution effectiveness and efficiency.

A feasible architecture is modeled as a graph AF which has the Hadoop ecosystem tools as nodes and the connections between the compatible interfaces of such tools as edges. Formally a feasible architecture is be modeled as follows.

Let ECO be the set of tools described in the tools description repository, $C = \{c|c \in ECO\}$ the subset of tools required by a feasible architecture, and $I = \{(c_i, c_j)|c_i, cj \in C\}$ the set of interfaces that connect the required tools. The feasible architecture AF is defined as:

$$AF = (C, I) \tag{8}$$

subject to:

$$\forall (c_i, c_j) \in I : c_i \neq c_j \tag{9}$$

$$\forall u, v \in C : \exists \mu = \langle u, v \rangle \tag{10}$$

The constraint in Formula (9) denotes the non-existence of loops or interfaces that connect a tool to itself. For its part, Formula (10) specifies that the graph AF is connected.

Notion of Affinity Between the Hadoop Ecosystem Tools and Problem Instances: Let $p \in P$ be a problem instance and $h \in ECO$ be a tool described in the tools description repository, where $P = \{p|p = (AG, RG, AR)\}$, RG is the set of general requirements of a problem p, $AR = \{ar_i\}$ the set of required activities by p. And $RE = \bigcup RE_i$ the union of every special requirement for each required activity ar_i.

Let $R* = RG \cup RE$ be the set of all requirements of a problem instance, $r = (r_f, r_v, r_w)$ with $r \in R*$ be any requirement.

Let $H = \{(h_f, h_v)\}$ be the set of ordered pairs made up with a tool feature h_f and its respective value h_v.

The measure of affinity between a problem instance requirement r and a tool h described in the repository is defined as the function $f : R* \times ECO \rightarrow \mathbb{Z}$, as follows:

$$f(r, h) = \begin{cases} r_w * \left[(h_v - r_v) + 1\right] & \text{si } h_v \geq r_v \wedge r_f = h_f \\ r_w * \left[h_v - r_v\right] & \text{si } h_v < r_v \wedge r_f = h_f \end{cases} \tag{11}$$

Based on the definition given in Formula 11, a measure of affinity between a problem p—in terms of its requirements $R*$—and a given tool h such as the function $F : P \times ECO \rightarrow \mathbb{Z}$ presented below:

$$F(p, h) = \sum_{r \in R*} f(r, h) \tag{12}$$

Feasible Architecture Generation Process: The process of generating feasible architectures aims to identify and interconnect a set of tools described in the repository that meet the requirements of a given problem with the highest possible affinity value.

Below are developed each one of the most relevant stages of this process:

Required Layers Identification: This first stage of the process aims to determine which layers are required to build a feasible architecture. This identification is done in function of the required activities that are described in the set $AR = \{ar_i\}$ defined in Formula (5) of the previously presented **problem model**. Formally, the set of required layers L is defined as:

$$L = \bigcup_{ar_i \in AR} \lambda(ar_i) \tag{13}$$

where $\lambda(ar_i) = \{l_i\}$ and l_i is a required layer by the required activity ar_i.

Tools Base Initialization for a Layer: In this phase, a base set of tools belonging to a layer is determined and the existence of at least one connection interface with the tools in the previously instantiated lower layers is validated. Formally, this set is defined as $H_{l_j} = \{h\}$ and it is subject to the following constraints:

Let H_{l_i} be the set of tools in the previously instantiated low level layer l_i, it is satisfied that:

$$\forall h \exists h' | h \in H_{l_j}, h' \in H_{l_i} : \exists \langle h, h' \rangle \tag{14}$$

Tools Weighting and Selection: For each one of the elements h inside the candidate tool base H_{l_i}, the measure of affinity with the requirements of the related problem $F(p, H)$ is computed. Then a selection of these tools that maximize the measure of affinity with the problem is performed, generating the subset of tools $H* \subseteq H$ with an optimal affinity value. Formally, this set is defined as follows:

$$H* = \left\{ h_i | h_i = \arg\max_{h \in H_{l_i}} F(p, h) \right\} \tag{15}$$

Feasible Architecture Instance Deployment and Evaluation: After determining the set of tools $H*$ for each required architectural layer, the following sequence of actions is performed:

1. Necessary connections are established between required interfaces.
2. Tools are enriched with the configuration recommendations and the related knowledge around its usage and implementation.
3. The affinity of the feasible architecture with the problem instance is evaluated. This evaluation is computed based on the affinities of the selected tools with the problem instance.

4 BiDArch: Design and Implementation

4.1 BiDArch Ontologies

BiDArch relies on a pair of ontologies to describe two domains of knowledge. In the first one, the Hadoop ecosystem toolkit repository is defined in terms of the features offered by tools. In the second one, the related knowledge to build a solution architecture is represented.

Figures 3 and 4 present the two ontologies previously mentioned. A high level abstraction model is shown with main classes and relationships for each of them.

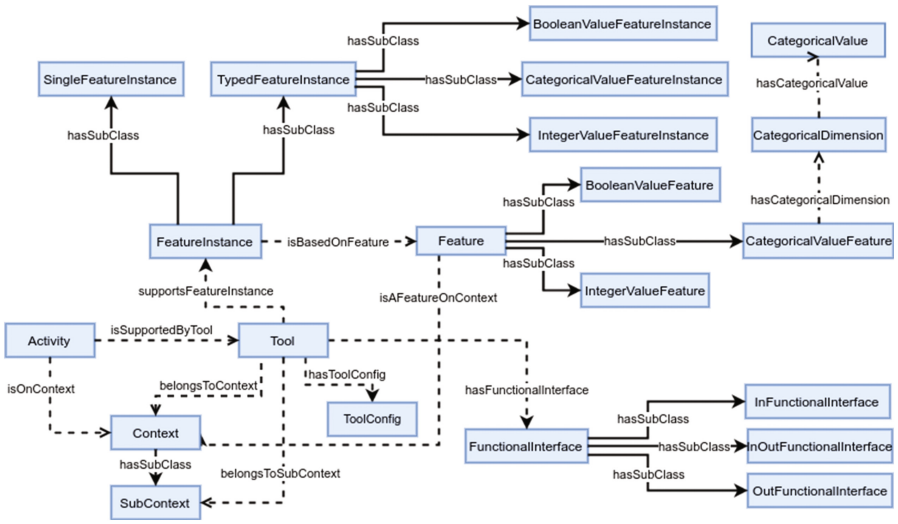

Fig. 3. Hadoop ecosystem toolkit repository ontology fragment.

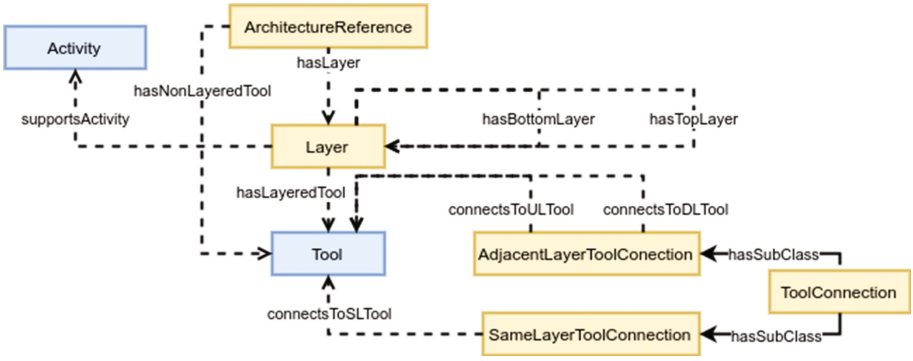

Fig. 4. Proposed reference model architecture ontology fragment.

4.2 BiDArch Architecture

The BiDArch Web application has a *client/server* architecture based on a three-layer model (data, process and presentation) as shown in Fig. 5. This application publishes a set of microservices that provides decoupled functionality and integrates information from various sources.

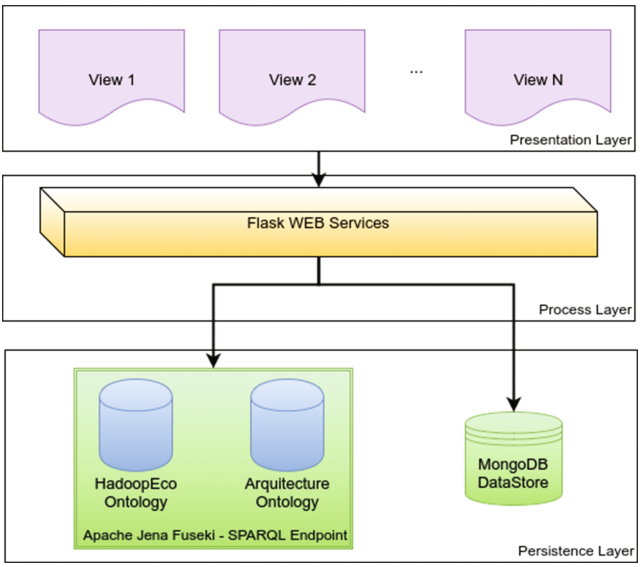

Fig. 5. BiDArch architecture.

BiDArch's user-defined problems instances are stored in a document-oriented database server instance, MongoDB [7]. Apache Jena Fuseki [8], a well-known SPARQL [9] *endpoint* stores the two application ontologies. Additionally, this server allows to set an inference engine or reasoner that is attached to the ontology, and in this way, triples that represent the additional knowledge are produced.

The application's business logic is deployed in Web services using the Web *microframework*, *Flask* [10]. It's done conceiving a future interoperability with external applications. In this Web application, many requests are received and attended to define a problem and build a solution feasible architecture.

The BiDArch application's frontend is intended to facilitate explicitly how an end user handles the characterization of a problem and dynamically supports his interaction with generated feasible solution architectures. To do this, the tool ExtJS 6.2 [11] is selected. This tool offers a large collection of graphics components with a high level of maturity. The problem specification process is supported by a wizard using a reversible sequence of steps. The visualization of the automatic built architectures is done using the open-source library, JointJS [12], taking advantage of the ease of use that it offers to model them as a graph.

5 Results

To show the obtained results in this paper, the following use case is presented:

The company NayroMan S.A.S, dedicated to the online marketing of bikes and spare parts, is interested in a software solution that allows to analyze its visitors' behavior while they are navigating on its concurrent website to browse product catalogs, compare bike parts and hopefully make a purchase. To do this, the company needs to ingest the web log activity during a long time, and then process and analyze the huge dataset obtained. Finally, the results need to be extracted to a traditional relational database management system to be used in its website.

This use case is based on the need for any architect to ingest and process data in a cluster and then extract the results to a relational database as Oracle. It was selected because it provides a composition of several required activities and represents a traditional scenario that can be associated with many problem instances.

BiDArch allowed to specify and build a feasible solution architecture for a real complete use case fully decomposed into several subproblems [13, pp. 74–84]. Each one of these five summarized problem instances are referenced below:

1. Data ingestion in a cluster [13, p. 75].
2. Batch processing after a previous data ingestion [13, p. 77].
3. Processed data extraction [13, p. 79].
4. ETL operations in a large dataset [13, p. 81].
5. Real time data querying in a large dataset [13, p. 83].

Tables 3 and 4 summarize the general requirements and specific required activities for a problem instance related to the presented use case respectively.

Table 3. General requirements for this problem instance.

Requirement	Weight	Value
Multilanguage	6	True
Fault tolerant	6	True
Security	7	True
Disk I/O usage	7	Low
Feature richness	7	High
Support level	5	High
Usability	7	Medium

BiDArch simplifies the definition of the problem model. The user is guided to select the main activities required to set a problem instance and the dependent related tasks are added automatically until Column 1 on Table 4 is complete.

Table 4. Requirements per activity for this problem instance.

Activity	Requirement	Weight	Value
Persistence	Data compression	5	True
	Strictly consistent writes	3	True
Ingestion	Filtering data to import	7	True
	Timeliness of data ingestion	5	Micro-batch
Batch processing	Job submission	5	True
	Automatic task execution optimization	7	True
	Resource manager independence	5	True
	Shared variables	3	True
Data extraction	Support for Oracle JDBC specific connectors	7	True
	Exporting with All-or-Nothing semantics	7	True
	Data copy validation	5	True
	Support for insertion and updates	5	True

Then, for each required activity, BiDArch suggests an indexed list of features that can be required, allowing to complete the requirements listed on Column 2. For each one of these requirements, the user is able to set up the importance updating the related requirement weight according to its particular problem's knowledge. Finally, a value of satisfaction for every feature is suggested and can be changed from a list of possible values in a specific domain.

Naturally, the richer a problem model is, the more effective the resulting selection of tools is. Therefore, the problem must be sufficiently well-defined regarding the general requirements and specific required activities to produce optimal feasible architectures. On the other hand, the definition of enough conditions to determine if every problem instance is well defined, is a very difficult task that can be done as future work.

BiDArch selects the tools highlighted on Table 5 that have the maximum value of affinity with the requirements of this problem instance.

Table 5. Selected tools with maximum affinity.

Layer	Activity	Tool	Affinity (units)
Persistence layer	Data storage	**HDFS**	**45**
		Apache HBase	18
Data movement layer	Data ingestion	**Apache Flume**	**57**
		Apache Kafka	15
		Apache Sqoop	35
Processing layer	Batch processing	**Apache Spark**	**58**
		Apache Hadoop MapReduce	5
Data movement layer	Data extraction	**Apache Sqoop**	**35**

The affinity calculations—presented on Table 5—are made using the notion of affinity expressed in Formula 11 and consuming the knowledge stored into the Hadoop ecosystem ontology. Additionally, the definition of more complex heuristics and processes to compute the affinity between tools and a problem instance can be very interesting. For example, a heuristic that considers the global impact of a group of features supplied by a particular tool, instead of the benefit obtained by individual features, can extremely reduce the complexity of generated feasible architectures.

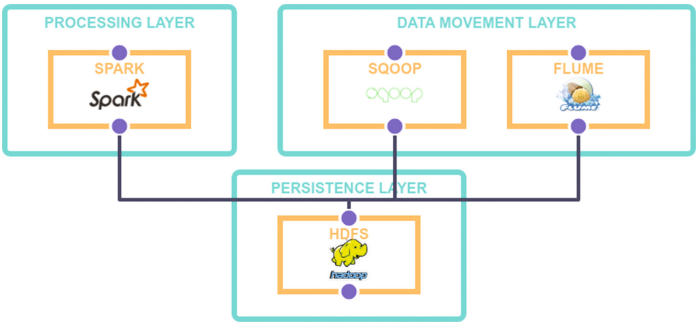

Fig. 6. Generated feasible solution architecture.

BiDArch automatically builds the feasible solution architecture shown in Fig. 6. Additionally, it provides an explanation to the question of why these tools were selected and evidences the notion of affinity they hold with the problem definition. It also enriches the architect's knowledge related to tools, showing him some parameters configuration suggestions and the available interfaces for integration with external tools.

6 Related Work

Big data players represent a set of large companies with the ability to overcome all the drawbacks and challenges facing many organizations interested in resolving big data problems. Among these companies are: Amazon AWS[1], IBM[2], HortonWorks[3], Cloudera[4], MapR[5]. The solutions provided by these companies focus in various use cases such as real-time analysis of security, risk detection and management in financial services, information flow processing, large datasets storage optimization, among many others. To do this, they use many ecosystem

[1] Amazon AWS, Inc. - Website: https://aws.amazon.com/.
[2] IBM, Inc. - Website: http://www.ibm.com/.
[3] Hortonworks, Inc. - Website: http://hortonworks.com/.
[4] Cloudera, Inc. - Website: https://www.cloudera.com/.
[5] MapR, Inc. - Website: https://www.mapr.com/.

tools such as Apache HBase [14], Apache Storm [15], Apache Hive [16], Apache Kafka [17] and obviously Apache Spark [18] among many others.

Some of these companies have their own Hadoop distributions with a range of additional features that represent differentiating values to their customers. Within these distributions of Hadoop stand out: HDP (Hortonworks Data Platform) [19], CDH (Cloudera Data Hub) [20] and MapR Data Platform [21].

Some experts on this area have made important publications to transmit their experience in the use of these tools and the lessons learned related to scenarios in which they are useful, as well as the way of how they should be used. For instance, the work [22] made by a group of Cloudera's members serves as a recommendation guide for the design Hadoop applications architectures and shows the implementation of some real use cases.

The project [2] presents an extensive list of tools that have some relationship with Hadoop ecosystem. The tools included in this repository are classified in several categories such as distributed file systems, multi-purpose distributed and scalable processing frameworks, NoSQL repositories and NewSQL repositories that aim to provide the same scalable performance of NoSQL systems for traditional relational database systems, among others.

As for reference architecture models, the Lambda architecture repository [23] consolidates a large number of electronic resources related to Lambda architecture real applications. This architecture was designed by Nathan Marz as a scalable and fault tolerant model for *big data* applications design. On the other hand, J. Kreps questions the Lambda architecture model in his article [24], exposing the weaknesses around the previous model and creating a new one that focuses on the possibility of offering the ability to develop, test, debug and operate solutions built on top of a simple processing framework.

7 Conclusions

This work presents the design and implementation of BiDArch, a feasible solution architectures generator for problem instances in *big data* domain. Based on a study and analysis of the documentation related to Hadoop ecosystem tools, we developed an ontology that describes the knowledge related to the tools' features.

The architectural requirements description process is performed according to the tools' features set on the previous ontology. This process is implemented using a three-step wizard that allows to represent three elements: a list of generic attributes, a set of general requirements that are related to transversal tools features and a set of requested activities. The particular requirements for any requested activity are defined in function of a specific set of features provided by a sub set of Hadoop ecosystem tools. This process is defined on top of two abstraction models: the **general requirement model** and the **problem model**. The first model encapsulates the definition of any requirement using the tool features and sets an order relationship between problem requirements. On the other hand, the problem model abstracts the three elements used to describe a problem instance.

With the tool presented in this work, we build feasible solution architectures for *big data* problems that require some traditional activities. These activities include: storage, ingestion, extraction, *batch* processing, ETL operations and querying on huge *datasets*. However, there are some specialized activities that need to be added, such as scalable processing on graphs and *machine learning*, which comprise an important domain of problem instances and enable high value solutions for some business targets.

References

1. Achari, S.: Hadoop Essentials - Tackling the Challenges of Big Data with Hadoop. Packt Publishing (2015)
2. Roman, J.: The Hadoop ecosystem table. https://hadoopecosystemtable.github.io/
3. IBM, Inc.: The four V's of big data. http://www.ibmbigdatahub.com/infographic/four-vs-big-data
4. Scott, J.: 7 key technologies in the evolving hadoop ecosystem. http://www.dbta.com/BigDataQuarterly/Articles/7-Key-Technologies-in-the-Evolving-Hadoop-Ecosystem-111505.aspx
5. Apache Spark MLlib. http://spark.apache.org/mllib/
6. Apache Mahout. https://mahout.apache.org/
7. MongoDB for giant ideas. https://www.mongodb.com/
8. Apache Jena Fuseki. https://jena.apache.org/documentation/fuseki2/index.html
9. W3C. SPARQL Query Language for RDF. https://www.w3.org/TR/rdf-sparql-query/
10. Flask (A Python Microframework). http://flask.pocoo.org/
11. Ext JS 6.2.0. http://docs.sencha.com/extjs/6.2.0/
12. JointJS: Visualize and interact with diagrams and graphs. https://www.jointjs.com/opensource
13. Sosa, J.: Generador de arquitecturas de solución sobre el ecosistema Hadoop para problemas de Big Data. https://webcat.uniandes.edu.co/uhtbin/webcat
14. Apache HBase. http://hbase.apache.org/
15. Apache Storm. http://storm.apache.org/
16. Apache Hive. https://hive.apache.org/
17. Apache Kafka. https://kafka.apache.org/
18. Apache Spark. https://spark.apache.org/
19. Hortonworks, Inc.: Hortonworks data platform. https://hortonworks.com/products/data-center/hdp/
20. Cloudera, Inc.: CDH is Cloudera's 100% open source platform distribution. https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html
21. MapR, Inc.: The MapR converged data platorm. https://mapr.com/products/mapr-converged-data-platform/
22. Grover, M., Malaska, T., Seidman, J., Saphira, G.: Hadoop Application Architectures, 1st edn. O'Reilly Media Inc., Sebastopol (2015)
23. Bijnens, N., Hausenblas, M.: Lambda Architecture. A repository dedicated to the Lambda Architecture (LA). http://lambda-architecture.net/
24. Kreps, J.: Questioning the Lambda Architecture. https://www.oreilly.com/ideas/questioning-the-lambda-architecture

http://www.springer.com/978-3-319-66561-0

Advances in Computing
12th Colombian Conference, CCC 2017, Cali, Colombia, September 19-22, 2017, Proceedings
Solano, A.; Ordoñez, H. (Eds.)
2017, XX, 807 p. 310 illus., Softcover
ISBN: 978-3-319-66561-0