Towards an Architecture for Managing Big Semantic Data in Real-Time*

Carlos E. Cuesta[1], Miguel A. Martínez-Prieto[2,3], Javier D. Fernández[2,3]

[1] VorTIC3 Research Group, Dept. of Comp. Languages and Systems II,
Rey Juan Carlos University, Madrid (Spain)
[2] DataWeb Research, Dept. of Computer Science,
University of Valladolid, Segovia & Valladolid (Spain)
[3] Dept. of Computer Science, University of Chile, Santiago (Chile)
carlos.cuesta@urjc.es, {migumar2,jfergar}@infor.uva.es

Abstract. Big Data Management has become a critical task in many application systems, which usually rely on heavyweight batch processes to process large amounts of data. However, batch architectures are not an adequate choice for the design of real-time systems, where expected response times are several orders of magnitude underneath. This paper outlines the foundations for defining an architecture able to deal with such an scenario, fulfilling the specific needs of real-time systems which expose big RDF datasets. Our proposal (SOLID) is a tiered architecture which separates the complexities of Big Data management from their real-time data generation and consumption. Big semantic data are stored and indexed in a compressed way following the RDF/HDT proposal; while at the same time, real-time requirements are addressed using NoSQL technology. Both are efficient layers, but their approaches are quite different and their combination is not easy. Two additional layers are required to achieve an overall high performance, satisfying real-time needs, and able to work even in a mobile context.

1 Introduction

Big Data is one of the buzzwords in the current technological landscape. A widely accepted definition says that Big Data is "*when the size of the data itself becomes part of the problem*" [11]. It basically states the most obvious dimension of Big Data: the **volume**. However, any definition is incomplete without considering **velocity** and **variety**. These *three V's* [8] comprises the most accepted Big Data characterization:

Volume: large amounts of data are gathered and stored in massive datasets created for different uses and purposes. *Storage* is the first scalability challenge in Big Data management since preserving the data must be our first responsibility. In turn, storage impacts in data *retrieval*, *processing* and *analysis*.

Velocity means how data flow, at high rates, in increasingly distributed scenarios. Two data streams can be distinguished: i) streams of new data (potentially generated in different ways and sources) being progressively integrated into existing (big)

* This work has been partially funded by the Spanish Ministry of Economy and Competitiveness through Projects TIN2012-31104, TIN2009-13838 and TIN2009-14009-C02-0; and also by Chilean Fondecyt Grant 1-110066, the Regional Government of Castilla y Leon and the ESF.

datasets, and ii) streams of query results (potentially large) to user requests. Thus, velocity means how fast data is produced, demanded and served.

Variety refers to various degrees of structure (or lack thereof) within the Big Data [9]. This dimension is motivated by the fact that Big Data may integrate multiple sources: *e.g.* any kind of sensor network, web server logs, politics, or social networks, among others. Obviously, each source describes its own semantics, resulting in different data schemas which are hardly integrable in a single model. Thus, Big Data variety demands a logical model enabling effective data integration.

These three V's provide a good description of Big Data, but the volume dimension must be revised from a practical perspective. Although one could think in terabytes, petabytes or exabytes talking about Big Data, few gigabytes may be enough to collapse an application running on a mobile device or even in a personal computer. Thus, the term Big Data is used, in this paper, to refer any dataset whose size is greater than the computational resources available for its storage and processing in a given system.

An architecture designed for managing Big Data [3] must consider all the dimensions above. However, its intended purpose restricts which one is first addressed. For instance, storage could be more critical for a mobile application than for another one running on a powerful server, whereas data retrieval speed is a priority for a real-time system but may not for a batch process. Thus, the dimensions must be prioritized in such a way that the resulting architecture covers effectively its requirements.

We design our proposal on a strong assumption: the more data are integrated, the more interesting knowledge may be generated, increasing the resulting dataset **value**. Thus, we *promote data variety over volume and velocity*. This decision is materialized through the model used for logical data organization. We choose a graph-based model because it can reach higher levels of variety before data become unwieldy. This allows more data to be linked and queried together [16]. The most practical trend, in this line, relies on the use of the Resource Description Framework (RDF) [12] as data model.

RDF is a cornerstone of the Semantic Web [5], whose basic principles are materialized by the emergent Web of Data. It reaches all its potential when it is used in conjunction with vocabularies providing data semantics, so we will use the term *big semantic data* to refer big RDF datasets. Note that choosing RDF as data model is a variety decision which does not discriminate any kind of software managing big semantic data. However, it restricts how these data are finally structured, stored, and accessed, so *it influences on volume and velocity dimensions*. Under this consideration, we design SOLID as a high-performance architecture for real-time systems consuming RDF data.

SOLID *(Service-OnLine-Index-Data architecture)* tackles this scenario through a tiered configuration which separates complexities of real-time data generation and Big Semantic Data consumption. On the one hand, the dataset must be preserved as compact as possible in order to save storage and processing resources. Thus, Big Data is stored and indexed in compressed way, enabling significative spatial savings and efficient query resolution (compression allows more data to be processed in main memory). These responsibilities are managed in the *Data* and *Index* layers, which are built around the RDF/HDT [13,7] features. However, these optimized representations are costly to update online. It means that, on the other hand, the pieces of RDF generated in real-time must be processed in a third layer which enables efficient data updates and also

provides competitive query resolution. This *Online* layer is implemented using NoSQL technology. Although both approaches are efficient by themselves, two additional layers are required to achieve an overall high performance: i) the *Merge* layer leverages RDF/HDT features for integrating the "online data" into the big semantic data; and ii) the *Service* layer supports efficient query resolution, by merging results retrieved from the Index and the Online layers using SPARQL [15].

The rest of the paper is organized as follows. Section 2.1 summarizes the current "RDF world". Section 3 describes the architecture SOLID, whereas Section 4 realizes it using different technologies from the state of the art. Finally, Section 5 concludes about our achievements and devises the SOLID evolution towards its final implementation.

2 Motivation

This section provides a brief review about *the RDF world* and introduces the main RDF features. More detailed information about this topic, and some examples about its current use, can be found in [10]. Then, we illustrate a use case highlighting the core requirements of managing big semantic data in practice.

2.1 The RDF World

The Resource Description Framework (RDF) [12] is an extremely simple data model in which an entity (also called resource) is described in the form (subject, predicate, object). For instance, describing a transaction (in the following example): `chek-in#1`, which was made the day `01/01/2013` in `station#123`, involves the triples: `(chek-in#1, date, "01/01/2013")` and `(chek-in#1, in_station, station#123)`. An RDF dataset can be seen as a graph of knowledge in which entities and values are linked via labeled edges. These labels (the predicates in the triples) own the semantic of the relation, hence it is highly recommendable to use standard vocabularies or to formalize new ones as needed.

RDF has been gaining momentum since its inception thanks to its adoption in diverse fields, such as bioinformatics, social networks, or geographical data. The Linked Open Data project plays a crucial role in the RDF evolution [10]. It leverages the Web infrastructure to encourage the publication of such semantic data [4], providing global identity to resources using HTTP URIs. Moreover, integration between data sources is done at the most basic level of triples, that is, to connect two data sources can be as easy as making connection between resources. For instance, a new triple: `(station#123, location, <http://dbpedia.org/page/Canal_Street>)`, enables all information stored in DBpedia[4] about *Canal Street* to be directly accessed from the aforementioned station.

This philosophy pushes the traditional document-centric perspective of the Web to a data-centric view, emerging a huge interconnected cloud of data-to-data hyperlinks: the *Web of Data*. Latest statistics[5] pointed that more than 31 billion triples were published and more than 500 million links established cross-relations between datasets.

It is worth noting that, although each piece of information could be particularly small (the *Big Data's long tail*), the integration within a subpart of this Web of Data

[4] DBpedia: `http://dbpedia.org`, is a partial RDF conversion of Wikipedia

[5] `http://www4.wiwiss.fu-berlin.de/lodcloud/state/` (September, 2011)

can be seen as big semantic data. RFID labels, Web processes (crawlers, search engines, recommender systems), smartphones and sensors are potential sources of RDF data, such as in our running example. Automatic RDF streaming, for instance, would become a hot topic, specially within the development of smart cities [6]. It is clear that Linked Data philosophy can be applied naturally to these *Internet of Things*, by simply assigning URIs to the real-world things producing RDF data about them via Web.

2.2 Running Example

Each time we use a metro card, the transaction is recorded. A register may include, at least, an anonymous identifier, the date and time of the transaction, and the checked station. Let us consider an information system storing and serving all this information for a city such as New York (NYC), having 468 stations, 5 millions rides per weekday and a total 1,640 millions rides per year[6]. It conforms a really interesting Big Data to query and analyze, subject to integration with other data (such as subway and city facilities, events or services), but increasingly growing minute by minute. After, five years of gathering, this huge information system will store 8,000 millions records and it has to be able to serve queries efficiently for a wide range of purposes, such as prediction and other logistic processes related to the metro management and NYC organization, statistics, business intelligence, etc. Under this scenario, thousand of records are generated every couple of minutes, which has to be integrated into the full knowledge base. On the one hand, it makes no sense to perform continuously insertions, as this would imply to regenerate the (huge) indexes to more than 8,000 millions records, suffering a significant performance degradation of the tasks running in the system. On the other hand, we could not afford an offline batch insertion (typically at the end of the day) as some of the processes, such as prediction, requires the latest records to perform real-time processes.

Let us describe how this decision applies to the previous example. We consider, for instance, that the check-in transaction is enhanced with data about breakdowns in other transports, meteorology or information about cultural and sporting events held in New York. These new data can be the glue to understand different displacement patterns, allowing better service plans to be adopted or increasing security in days of massive traffic, among other possible decisions. Although it is out of the scope of this paper, the first step is to design a specific vocabulary for transaction modeling. Once the vocabulary is agreed, each transaction is represented as a small piece of RDF which must be integrated into the dataset. Thus, a competitive throughput must be provided to integrate all these pieces of data since no transactions may be lost. Besides, continuous data streams of breakdowns, weather or events are also received, but these additional data can be considered as "small data", so we can process them efficiently. Moreover, all the data must be available for querying and these requests must be resolved efficiently.

3 The SOLID Architecture

Our solution for the stated problem –to be able to reconcile the requirements of Big Data processing and the need to consider the flow of incoming real-time information–

[6] NYC statistics 2011, `http://mta.info/nyct/facts/ridership/index.htm`

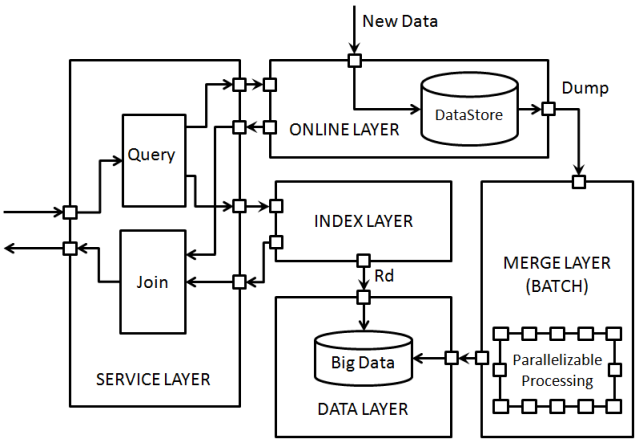

Fig. 1. Layered configuration of the SOLID architecture.

is to define a generic architecture able to simultaneously deal with both approaches. The problem could seem rather specific, but it actually applies to a large set of applications; indeed, there are even some architectural approaches which tackle a similar problem, such as Lambda [14]. Our proposal, however, emphasizes the use of semantic technologies, and has a different setting. Apart from that, this architecture has been designed not to depend on any particular domain or technology.

Our proposal consists of a specific architecture, named SOLID after the enumeration of its main components: *Service-OnLine-Index-Data*, which gathers lessons learned from batch architectures, real-time overlays, and high-performance semantic datastores. It can be described as a 3-tier architecture, in which the middle tier is also subdivided in three layers (*OnLine*, *Index* and *Data*), defining a multi-tiered, layered architecture. It is summarized in Fig. 1, and its five layers are quickly mentioned in the following.

Online Layer. The top layer of the architecture, which is also the one which specifically deals with real-time needs. It captures the incoming flow of new data in a high-speed datastore, which is used for temporary storage. It works as a large buffer for real-time incoming data: what is not (still) in the main database is located here.

Data Layer. The bottom layer of the architecture, which contains the main datastore, *i*.e. the Big Data repository. It is designed as a storage layer, able to maintain large sets of information, their organization and their semantics - but it needs not to be inherently fast, just self-descriptive.

Index Layer. The middle layer, which provides an *index* for the Data Layer, therefore turning it into a high-speed datastore. It is built using the semantic metadata in the lower layer, and designed to provide a fast access to the information it contains: all queries to the main datastore are performed using this index. This way we are able to maintain a large, stable datastore with a quick access.

Service Layer. The most external tier, which presents a façade to the outside user. All queries are performed through this layer, which multiplexes them adequately, and forwards it to both the Online and Index Layers. Each layer provides the corresponding result set; these are joined and combined to provide an unified answer. The internal

modules in this layer can therefore be described as mediators, and have themselves a pipe-filter structure.

Merge Layer. The most internal tier, which provides one of the key steps of our approach. In some chosen moment, the Online Layer *dumps* the contents of its datastore. The Merge Layer receives and transforms this input and combines it with existing data, producing a fresh copy of the Big Data store, which is fed into the Data Layer, without altering its structure, *i.e.* maintaining immutability properties. This process requires much computational effort, so it is performed in batch as a massively parallel process using, for instance, MapReduce-based computation.

Therefore, in summary: the data architecture uses a large datastore for Big Data, which is indexed to allow a fast access. At the same time, the flow of real-time data is stored in a temporary datastore, until it is eventually merged in the main store by using a massive-parallel batch process. When the architecture receives a query, it is forwarded to the two datastores, and the corresponding result sets are merged to provide a federated answer, in a completely user-transparent way.

This "abstract" architecture has to make certain decisions to be applied to a concrete case - such as the nature of the datastores, or the different implementations. The next section is devoted to explain this sort of compromises.

4 SOLID in Practice

Once we have characterized the SOLID layers, we are able to devise its implementation. First, we analyze practical decisions made around the data-centric layers: *Data*, *Index*, and *Online*, which are designed by leveraging possibilities of binary RDF. Then, we outline *Merge* and *Service* layers, emphasizing how their processes can also take advantage of binary RDF for improving performance.

4.1 Data-centric Layers

One main decision when managing RDF data is the underneath format, since the RDF data model does not restrict its concrete representation. RDF/XML [2] was originally recommended as the RDF syntax. It restructures the RDF graph to be encoded as a XML tree, preserving a document-centric view which is not acceptable when big semantic data must be serialized. The potential of huge RDF is seriously underexploited due to the large space they take up [13]. Some other syntaxes has been proposed later [10], but RDF/HDT[7] [13,7] was the first designed bearing in mind final serialization sizes.

RDF/HDT is a binary syntax which reduces verbosity in favor of machine-understandability. It allows Big Semantic Data to be efficiently managed within the common workflows of the Web of Data, being an ideal choice for storage and transmission (it takes up to 15 times less space than traditional RDF syntaxes [7]). In addition, the RDF/HDT specification comprises specific configurations of compact data structures which enable to easily parse and load Big Semantic Data in compressed space.

The **Data Layer** stores the Big Semantic Data serialized in RDF/HDT. It enables large spatial savings and guarantees data immutability in compressed space. Besides,

[7] HDT is W3C Member Submission from 2011: `http://www.w3.org/Submission/2011/03/`.

it allows big datasets to be efficiently mapped to the memory hierarchy by using the RDF/HDT data structures. This feature is the SOLID core for accessing Big Semantic Data because any RDF triple can be retrieved without prior decompression. Thus, this layer assumes the complexity of Big Data storage and set the basis for the Index Layer.

The **Index Layer** exploits the Data Layer features for supporting efficient query resolution over the Big Semantic Data storage. It implements the HDT-FOQ (HDT *Focused on Querying*) proposal [13] which, basically, builds two lightweight indexes on top of the RDF/HDT representation. These structures enable fast lookups over the Data Layer, and it results in efficient SPARQL resolution reporting competitive performance with respect to the state of the art of RDF stores [13].

The **Online Layer** assumes the complexity of managing data generated in real-time, keeping the Data Layer immutable. In fact, this philosophy perfectly fits the lower layers, as RDF/HDT was thought to be read-only and updates are costly. On the one hand, the Online Layer must allow fast write operations in order to insert all new generated data. On the other hand, it must resolve SPARQL in an efficient way, although scalability issues are minimized in this case by considering that this Layer only stores a small subset of data (for instance, in the running example, the check-in transactions generated in the current day). This layer implementation depends on the computational resources available. We suggest to use general-purpose NoSQL technology or any native RDF store, because these report the best numbers for managing RDF [1].

4.2 Processing-centric Layers

These two layers play intermediary roles between the data-centric layers. Both ones leverage RDF/HDT features for implementing their processing tasks.

The **Merge Layer** implements the batch process which merges the Big Semantic Data (from the Data Layer), and the data recorded in the Online Layer. It is a time-consuming process which requires a high-performance configuration. As explained, it can be realized using Map-Reduce. Whereas the Big Semantic Data from the Data Layer is already in RDF/HDT, the data dumped from the Online Layer must be firstly converted to RDF/HDT; this conversion can take place in the Merge Layer, or it can be held in the Online Layer. In any case, the new Big Semantic Data is not obtained from the scratch because the process leverages internal RDF/HDT ordering for efficient merging. It results in a cost proportional to the smallest dataset size, isolating possible Big Data drawbacks. This process finishes delivering a new Big Semantic Data representation which replaces the previous one in the Data Layer. This fact triggers two additional operations: i) the Index Layer updates its structures according to the new data and ii) the Online Layer clears all data currently integrated into the Data Layer.

The **Service Layer** relies on the SPARQL expressiveness to satisfy all possible queries. Its inner pipeline first duplicates the corresponding query and sends the copies to the filters responsible for interacting with the Service and Index Layers. Each layer resolves the query independently and delivers its results to the last filter, which finally computes the results. In practice, results from the Service Layer are first obtained, and then there is room for optimization. They can be processed to obtain a hash structure which allows to efficiently join both result sets before final delivery.

5 Conclusions and Future Work

This paper proposes SOLID, a new architecture for managing Big Semantic Data in real-time. It is designed upon the property of complexity isolation allowing the main concerns to be resolved independently. On the one hand, we rely on binary RDF/HDT features for storing and indexing RDF in compressed space. On the other hand, we use NoSQL technology for recording real-time data. The overall dataset is queried through a pipeline-based layer which asks to the big and online representations, retrieves results, and then merges them for delivering. Online and Big Data are merged programmatically by performing a parallel process: *e.g.* Map-Reduce.

The empirical results reported by each technology considered in SOLID bodes well for our future work. We are currently implementing the different layers and their interfaces to obtain an optimized prototype for servers. Besides, we are working on a lightweight SOLID version designed for mobile devices because our spatial savings enable more data to be processed in these limited devices. Nevertheless, their practical requirements are different and, for instance, the merge layer is discarded because its functionality is not directly required.

References

1. D. Abadi, A. Marcus, S. Madden, and K. Hollenbach. Scalable semantic Web data management using vertical partitioning. In *Proc. of VLDB*, pages 411–422, 2007.
2. D. Beckett, editor. *RDF/XML Syntax Specification*. W3C Recommendation, 2004.
3. E. Begoli and J. Horey. Design Principles for Effective Knowledge Discovery from Big Data. In *Proc. 2012 Joint WICSA/ECSA Conference*, pages 215–218. IEEE, Aug. 2012.
4. T. Berners-Lee. *Linked Data: Design Issues*. 2006. `http://www.w3.org/DesignIssues/LinkedData.html` (retrieved on 2013-03-01).
5. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 2001.
6. S. De, T. Elsaleh, P. Barnaghi, and S. Meissner. An Internet of Things Platform for Real-World and Digital Objects. *Scalable Computing: Practice and Experience*, 13(1), 2012.
7. J. Fernández, M. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF representation for publication and exchange (HDT). *Journal of Web Semantics*, 2013. In Press. `http://dx.doi.org/10.1016/j.websem.2013.01.002`.
8. Y. Genovese and S. Prentice. Pattern-Based Strategy: Getting Value from Big Data. Gartner Special Report, June 2011.
9. A. Halfon. *Handling Big Data Variety*. `http://www.finextra.com/community/fullblog.aspx?blogid=6129` (retrieved on 2013-03-01).
10. T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 2011.
11. M. Loukides. Data Science and Data Tools. In *Big Data Now*, chapter 1. O'Reilly, 2012.
12. F. Manola and E. Miller, editors. *RDF Primer*. W3C Recommendation, 2004.
13. M. Martínez-Prieto, M. Arias, and J. Fernández. Exchange and Consumption of Huge RDF Data. In *Proc. of ESWC*, pages 437–452, 2012.
14. N. Marz and J. Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning, 2013.
15. E. Prud'hommeaux and A. Seaborne, editors. *SPARQL Query Language for RDF*. W3C Recommendation, 2008. `http://www.w3.org/TR/rdf-sparql-query/`.
16. R. Styles. *RDF, Big Data and The Semantic Web*. `http://dynamicorange.com/2012/04/24/rdf-big-data-and-the-semantic-web/` (retrieved on 2013-03-01).