A Resource Oriented Architecture to Handle Data Volume Diversity

Pierre de Vettor, Michael Mrissa, Djamal Benslimane

A Resource Oriented Architecture to Handle Data Volume Diversity

Pierre De Vettor[1], Michaël Mrissa[1], and Djamal Benslimane[1]

Université de Lyon, CNRS
LIRIS, UMR5205, F-69622, France
Lyon, France
firstname.surname@liris.cnrs.fr

Abstract. Providing quality-aware techniques for reusing data available on the Web is a major concern for today's organizations. High quality data that offers higher added-value to the stakeholders is called smart data. Smart data can be obtained by combining data coming from diverse data sources on the Web such as Web APIs, SPARQL endpoints, Web pages and so on. Generating smart data involves complex data processing tasks, typically realized manually or in a static way in current organizations, with the help of statically configured workflows. In addition, despite the recent advances in this field, transfering large amounts of data to be processed still remains a tedious task due to unreliable transfer conditions or transfer rate/latency problems. In this paper, we propose an adaptive architecture to generate smart data, and focus on a solution to handle volume diversity during data processing. Our approach aims at maintaining good response time performance upon user request. It relies on the use of RESTful resources and remote code execution over temporary data storage where business data is cached. Each resource involved in data processing accesses the storage to process data on-site.

Keywords: resource oriented architecture, data integration, data semantics, smart data

1 Introduction

During the last few years, governments, companies and organizations have opened their databases and information systems to the world across the Web, thanks to initiatives such as the open data project [8]. These data sources are typically exposed via Web APIs [11] or SPARQL endpoints and can be combined in service mashups [1] to produce highly valuable services. As an/For example, the sets of APIs provided by Twitter, Amazon, Youtube or Flickr are reused/used again in thousands of mashups[1].

This smart use of data has caught the interest of the community as a natural development. The objective of smart data [13] is focused on producing high-quality data that is directly useful to users. Automatically integrate data from

[1] See also http://www.programmableweb.com/

diverse sources in order to produce smart data is currently a hot research topic. Despite these advances, data quantity still hampers data exchanges and remains a bottleneck in architectures, especially when it comes to data transfer between resources. By taking advantage of REST architectural style and the use of resources, we unfortunately suffer from the limitations of Web protocols, and it is sometimes difficult to transfer large quantities of data through HTTP. There is a need for an approach that minimizes data transfer and performs data processing tasks closer to the data source to reduce network traffic.

In this paper, we present our adaptive architecture and propose a solution, through the use of adaptive data access strategies and remote code execution on temporary data storage unit resources, to handle latency and architecture issues that appear when processing data volume diversity. Our architecture optimizes and adapts workflows to handle the variety of data sources involved in the query. In this approach, we present this resource oriented architecture and explain our solution for reducing latency in resource oriented architecture.

This paper is organized as follows. Section 2 presents related approaches to handle volume diversity in data sources in RESTful architectures. Section 3 presents our resource oriented architecture and our solution to minimize data exchange. Section 4 gives an evaluation of our prototype in terms of responsiveness and shows how it responds to user requests with acceptable timings. Section 5 discusses our results and provides guidelines for future work.

2 Related Work

Over the past few years, big data has generated a lot of interest from researchers and industrials, answering to four challenges, known as the four Vs: Volume, Variety, Velocity and Veracity. Defining our smart data architecture [3], we provide data source models and strategies to support the Variety challenge and analysis, combination and data cleaning for Veracity. Finally, Velocity is managed by adapting workflows and optimizing resource orchestration at runtime to improve response time. On top of that, we put more focus on data quality through the use of semantics, metadata and intelligent processing techniques. In this smart data context [13], propositions focus on data quality rather than quantity and build smarter architecture. In this data-driven context, Web standards comes as a solution, but limits data exchanges. The following appproaches have addressed the data issue in HTTP-based solutions.

Devresse et Al. [4] propose a library called *libdavix* allowing high performance computing world to benefit from HTTP and the RESTful principles. This approach focus on adapting the HTTP protocol, maximizing the reutilization of TCP connections, by providing a dynamic connection pool coupled with the usage of the HTTP Keep-Alive feature. By avoiding useless protocol handshakes, reconnections and redirections, their approach improves efficiency of large data transport through HTTP. In *Fast Optimised REST (FOREST)* [9], Ko et Al. propose a non-invasive technique relying on TCP data encapsulation in UDP-based data transfer payloads [6]. Evaluations shows good results, but the ap-

proach does not seem to provide a real solution, it is a low level fixing to benefit from advantages of other protocols. Zheng et al. [15] provide an overview of service-generated big data as well as big data-as-a-service, a flexible infrastructure providing common big data management functionalities. Their approach rely on cloud computing techniques to handle collection, storing, processing and visualization of data and they address some significant challenges, particularly about variety or volume and how infrastructure must support (and adapts) this variety and volume to provide fast data access and processing. Van Der Pol et Al. [14] propose an approach based on multiple paths TCP [5] to transfer huge data sets over networks. Their approach relies on load balancing transfer through the different available paths relying on parallel multiple TCP requests. Their approach can handle different paths with different bandwidths, balanced over the different interface offered by the system. They propose a prototype According to these approaches, it becomes clear that the most powerful solution is to minimize data transfers, process data volumes closer from the source to reduce the unnecessary traffic. In the next section, we present our resource-oriented architecture, the models that helps to build it, and finally, we present our solution to handle data volume diversity in our smart data architecture.

3 Contribution

In order to handle our smart data challenges, we envision a resource-oriented architecture, generating adaptive workflows at runtime to adapt to data source characteristics. We rely on the data source models presented in our previous work [3] to represent data source characteristics such as uri, request format, volume, latency, etc. Relying on these models, we are thus able to generate adaptive *data processing workflows* according to characteristics that appears on data source descriptions.

Then we define an resource oriented architecture to manage these adaptive workflows and the resources involved.

3.1 Architecture

In our approach, we define different necessary steps to complete the data aggregation process and produce smart data. The main steps are **extraction** from data source and transformation into a pivot format, **semantic annotation** of the extracted data set ([2], [7]), **combination** [12] of obtained data sets, **filtering** of data sets in order to remove inconsistent or duplicated data. We divide each of these tasks as RESTful resources in our architecture and orchestrate them as workflows. In order to handle data flows between resources, we define an orchestrator, which acts as a data bus. This orchestrator forwards messages and data from and to resources, builds HTTP request and handles requests responses. Fig. 1 shows the different resources and components of our architecture. On top of that, we provide our architecture with management API and resources to avoid manual configurations as much as possible.

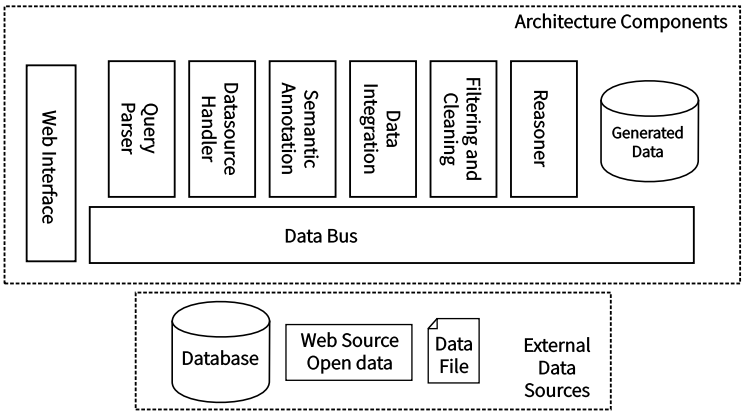

Fig. 1. Our Resource Oriented Architecture

3.2 Handling Volume Diversity

In order to handle volume diversity in our RESTful architecture, and to maintain a good response time performance, we propose a solution which reduces data transfer to a required minimum.

In our architecture, upon query request, an adapted workflow is generated, involving resources and services, to handle the required processing tasks. This workflow is executed, our data bus handles data exchanges, generating HTTP requests and retrieving responses from resources. We identified the following data transfers during the process: extraction from data sources, data transfer between core resources, and data download by the user. In this context, data extraction and download are required, but transfer between resources can be avoided or modified.

Based on this observation, we modify our architecture to decrease the volume of exchange data between resources. We design our API to only manipulate queries and metadata (data models, etc.). Computing this metadata, remote processing codes are generated in order to complete the process managed by this resource. These processes are handled close to data, minimizing execution time by lowering latency and network time.

3.3 Storing data

In order to *temporarily* store data, our proposal is based on temporary data storage units, generated at each user request. Storage units act as file hosting services, they are generated at runtime, for each *new* user request, for a limited amount of time and contains the data and metadata for this request. They are provided with an engine capable of processing remote processing tasks generated by resources. Storage units are erased after a certain amount time, which has been fixed to a default value of 24 hours. This delay is customizable for each

request. Time counter is reset each time a user reissues the query or reaccesses the storage. Storage units are accessible as RESTful resources, for management purpose or to retrieve query responses when data processing is over. When the processing tasks are over, the storage URI is given to the user, so that he could download the data sets answering to his request.

3.4 Processing engine

In order to handle the different tasks required to complete the smart data process, we provide the storages with a functional engine capable of executing remote processing tasks directly above the data instances. These codes are generated by the tasks resources and transfered instead of data.

We rely on functional languages to define processing tasks, since our data sets are stored as tabular data sets (lists or dictionary in JSON or JSON-LD[10]). We provide data store with different processing engines, each representing an environment or an engine. Each data store is provided with an API, which allows its management, but also to register engine libraries or plugins, required to process the different langages or functionnalities This data store is provided with an API, allowing to register different engine libraries or plugin to handle specific languages or functionnalities.

4 Implementation and Evaluation

In this paper, we focus our work on handling data volume diversity in our architecture for smart data management.

In order to evaluate the scalability of our architecture, we demonstrate the evolution of performance, evaluating request response time when answering to a set of complex semantic queries over multiple data sources. We vary the number of data sources and measure response times.

4.1 Use Case Scenario

We focus our work on the enrichment and reusability of data, and based our models and implementation on the data handled by a communication company, which has a need for an adaptive system able to automatically refine and combine data coming from their internal information system and enrich it with help from open Web data sources. This solution has for objectives to study the impact of campaign broadcasts over a list of customers and to provide decision support tools for future broadcasts. This scenario describes the following data sources, presenting several characteristics, specific to our scenario:

Source 1 is a linked service giving access to our company small business data set. Data that come from this source are subject to privacy constraints. **Source 2** is a SQL endpoint to a database that contains millions of tuples with a low update frequency. **Source 3** is another SQL endpoint to a database that contains customer daily activities, updated regularly. **Source 4** is a RSS

stream that contains user update requests (removal requests from customers, request form architecture's user). Other **sources** are represented by a set of Web sources: *FOAF* and *vcard* ontologies that help to annotate data, as well as a Dbpedia SPARQL endpoint. Relying on the scenario presented above, we create two request, involving different concepts subgraphs. We populate our scenario with a set of data sources, covering the different subgraphs.

```
PREFIX al: <http://restful.alabs.io/concepts/0.1/>
PREFIX xsd: <http://www.w3.org/TR/xmlschema-2/>
SELECT ?email_value ?campaign WHERE {
    ?email  a  al:email ;
            al:has_email_value  ?email_value ;
            al:blacklist_status ?status .
    ?clic  a   al:clic ;
           al:clic_email       ?email ;
           al:clic_date        ?date .
    FILTER  (?status != 1 && ?date >= "1411477450"^^xsd:date)
```

Listing 1.1. Query 1.1 involving the different concepts

```
PREFIX al: <http://restful.alabs.io/concepts/0.1/>
SELECT ?email_value WHERE {
    ?email      a                   al:email ;
        al:has_email_value  ?email_value ;
        al:blacklist_status ?status .
    FILTER  (?status != 1)
}
```

Listing 1.2. Query 1.2 introducing a user specific filters

Query 1.1 involves four subgraphs, implying data sources with different characteristics such as high volume (scenario's big database) and privacy sensitive information (scenario linked service). This query also presents user specific filters. Query 1.2 involves only a few number of concepts, and present less data manipulations. Tests are performed on a double core 2.3 GHz machine, with 4 GB of RAM. Restful resources and architecture are implemented through PHP frameworks, Zend and Slim[2] and hosted on scenario company servers (Apache).

4.2 Evaluation

We evaluate our architecture response time to query $Q1$, and $Q2$ respectively, when the number of data sources increases. We compared response time evolution for the same queries and data, with and without our optimisation process.

As can be seen in Fig. 2 and Fig. 3, the volume of data increase with the number of data source, and the classical approach response time suffer, due to HTTP transfers. In parallel, optimized approach response time increase linearly, but stays acceptable standing under the treshold of 4 seconds. In this case, transfer and network latency consumes time exponentially. Query $Q2$ involves less concepts, and as a result less data manipulations, but our architecture still suffer from a very high latency due to data transfer. When the number of data source exceed 24, the classical approach is unable to provide a response in an acceptable delay. This graph clearly show that our architecture can adapt to large volumes of data, using our optimisation technique.

[2] See http://www.slimframework.com/

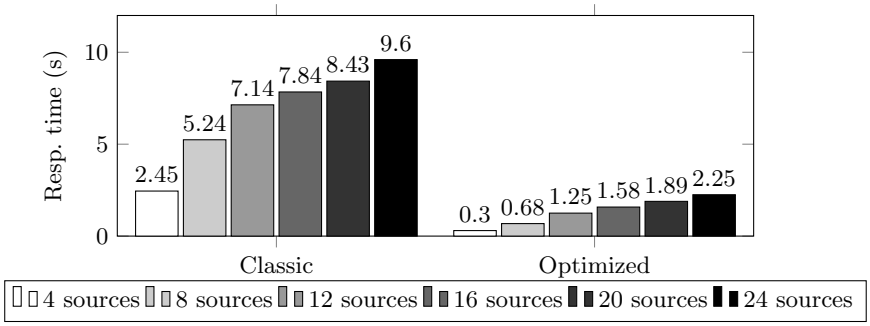

Fig. 2. Evaluation of average response time for query 1.1

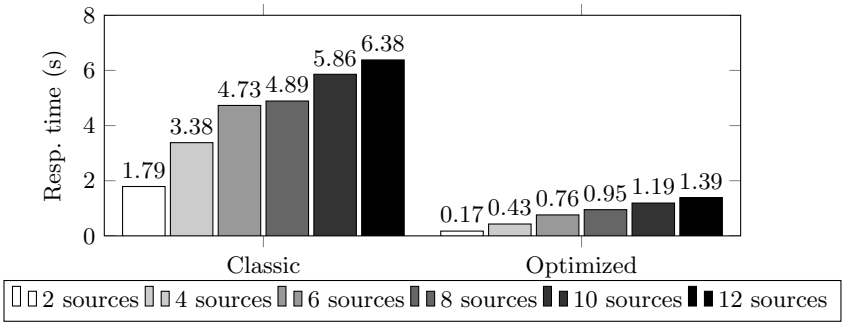

Fig. 3. Evaluation of average response time for query 1.2

5 Conclusion

In this paper, we describe a resource-oriented architecture that relies on adaptive data processing strategies to optimize data exchanges between resources that process data. We focus on the latency problems that appear when dealing with diverse data volumes especially when transferring data between the different architecture components. All along the smart data construction process, we rely on a temporary data storage where business data is stored. Our architecture handles the communication between the different resources, by transferring metadata about the query and the data storage URI. Resources generate adapted remote processing codes, which are forwarded to storage units and executed on-site by the data storage engine. Therefore, data manipulation is performed on-site, and the data does not flow through the architecture. By reducing latency due to data transfers, we alleviate our decentralized and distributed architecture from the burden of data tranfer, and improve the responsiveness of data-driven resource workflows. We support our implementation with a set of evaluations and tests through our use case scenario. As future work, we envision to investigate the appearance of data uncertainty in data aggregating approach, relying on probabilistic integration techniques.

References

1. Djamal Benslimane, Schahram Dustdar, and Amit P. Sheth. Services mashups: The new generation of web applications. *IEEE Internet Computing*, 12(5):13–15, 2008.
2. Christian Bizer. D2rq - treating non-rdf databases as virtual rdf graphs. In *In Proceedings of the 3rd International Semantic Web Conference (ISWC2004*, 2004.
3. Pierre De Vettor, Michaël Mrissa, and Djamal Benslimane. Models and architecture for smart data management. Technical Report RR-LIRIS-2014-017, LIRIS UMR 5205 CNRS/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/École Centrale de Lyon, December 2014.
4. Adrien Devresse and Fabrizio Furano. Efficient HTTP based I/O on very large datasets for high performance computing with the libdavix library. *CoRR*, abs/1410.4168, 2014.
5. A Ford, C Raiciu, M Handley, S Barre, and J Iyengar. Architectural guidelines for multipath tcp development. In *IETF, RFC 6182*. Citeseer, 2011.
6. Robert L. Grossman, Yunhong Gu, Xinwei Hong, Antony Antony, Johan Blom, Freek Dijkstra, and Cees de Laat. Teraflows over gigabit {WANs} with {UDT}. *Future Generation Computer Systems*, 21(4):501 – 513, 2005. High-Speed Networks and Services for Data-Intensive Grids: the DataTAG Project.
7. Lushan Han, Tim Finin, Cynthia Parr, Joel Sachs, and Anupam Joshi. Rdf123: From spreadsheets to rdf. In *The Semantic Web - ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 451–466. Springer Berlin Heidelberg, 2008.
8. Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.
9. R.K.L. Ko, M. Kirchberg, Bu-Sung Lee, and E. Chew. Overcoming large data transfer bottlenecks in restful service orchestrations. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 654–656, June 2012.
10. Markus Lanthaler and Christian Gutl. On using json-ld to create evolvable restful services. In *Proceedings of the Third International Workshop on RESTful Design*, WS-REST '12, pages 25–32, New York, NY, USA, 2012. ACM.
11. M. Maleshkova, C. Pedrinaci, and J. Domingue. Investigating web apis on the world wide web. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pages 107–114, Dec 2010.
12. Michael Mrissa, Mohamed Sellami, Pierre De Vettor, Djamal Benslimane, and Bruno Defude. A decentralized mediation-as-a-service architecture for service composition. *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 0:80–85, 2013.
13. Amit Sheth. Transforming big data into smart data: Deriving value via harnessing volume, variety, and velocity using semantic techniques and technologies. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 2–2, March 2014.
14. R. van der Pol, S. Boele, F. Dijkstra, A. Barczyk, G. van Malenstein, J. H. Chen, and J. Mambretti. Multipathing with mptcp and openflow. *High Performance Computing, Networking Storage and Analysis, SC Companion:*, 0:1617–1624, 2012.
15. Zibin Zheng, Jieming Zhu, and M.R. Lyu. Service-generated big data and big data-as-a-service: An overview. In *Big Data (BigData Congress), 2013 IEEE International Congress on*, pages 403–410, June 2013.