Towards a Scalable Cloud-based RDF Storage Offering a Pub/Sub Query Service

Laurent Pellegrino, Françoise Baude, Iyad Alshabani
INRIA-I3S-CNRS, University of Nice-Sophia Antipolis
2004 route des lucioles, Sophia Antipolis, France
lpellegr@inria.fr, fbaude@inria.fr, ialshaba@inria.fr

***Abstract*—Recently, Complex Event Processing engines have gained more and more interest. Their purpose consists in combining realtime and historical data in addition to knowledge bases to deduce new information. Also, RDF is now commonly used to make information machine-processable. In this short paper we propose to leverage existing research about distributed storage and realtime filtering of RDF data with the intention of helping Complex Event Processing engines to reach their goal at large scale. Towards this objective, we identify and discuss the challenges that have to be addressed for providing a solution that supports RDF data storage and a pub/sub retrieval service on a cloud-based architecture. Then, we explain how to meet these challenges through a solution based on structured Peer-to-Peer networks. Finally, we discuss the status of our ongoing work whose implementation is realized thanks to ProActive, a middleware for programming cloud-based distributed applications.**

***Index Terms*—RDF (Resource Description Framework); Publish/Subscribe; RDF data management; Cloud Computing.**

I. Introduction

In the past years, Cloud Computing gained a great interest in academic and industrial solutions. Its goal is to provide users with more flexible services in a transparent way. All services and applications are allocated in the cloud which is an internet-scale collection of connected devices. Aside, the exponential growing of information exchanged over the internet leads to the emergence of the Semantic Web [1] whose realization is brougth into existence thanks to RDF (Resource Description Framework) [2]. RDF is a W3C standard aiming to improve the World Wide Web with machine processable semantic data. It provides a powerful data model for structured knowledge representation and is used to describe semantic relationship among data. Statements about resources are in the form of $(subject, predicate, object)$ expressions which are known as *triples* in the RDF terminology (each element of a triple is dubbed RDF term). The subject of a triple denotes the resource that the statement is about, the predicate denotes a property or a characteristic of the subject, and the object presents the value of the property. RDF is increasingly used due to its interoperability [3], its good properties in data exchange and its potential use of inferencing to contextually broaden search, retrieval and analysis.

The traditional way of querying RDF data is a blocking *get* operation. However, applications need an asynchronous query mode to be more responsive on arrival of RDF data. Publish/subscribe (pub/sub) is a messaging pattern where publishers and subscribers communicate in a loosely coupled fashion. Subscribers can express their interests in certain kinds of data by registering a subscription (continuous query) and be notified asynchronously of any information (called an *event*) generated by the publishers that matches those interests. Notifications are made possible thanks to a matching algorithm that puts in relation publications and subscriptions.

Our goal is to provide a system, deployed in a cloud environment, that stores RDF events persistently, filter and notify them as soon as they arrive. For example, Complex Event Processing (CEP) [4] systems have a need to mix real-time, past events and existing knowledge bases to deduce new patterns [5]. However, the system we envisage is not limited to the integration with CEP engines. More generally, it could be used to take advantage of its distributed storage and pub/sub layer.

This short paper identifies some challenges that have to be addressed in order to build a distributed system that combines RDF data storage and pub/sub. In Section II, we highlight some of the challenges to take up when this type of system has to be built. Section III motivates and defines our retrieval model. Section IV explains how we expect our system to meet the challenges in line with our model, and how it differs from existing systems. Section V presents our conclusions.

II. Challenges

Proposing solutions for or against the following requirements or difficulties constitutes the challenges:

1) Scalability: In our context, a scalable system must be able to support a large number of data, publications and subscriptions. This is the key property to fulfill when a distributed system is built. Also, in pub/sub systems, expressivity and scalability are closely related [6], [7]. Expressivity implies that events with different formats and semantics are supported in addition to a powerful subscription language (i.e. a subscription language that offers the possibility to consumers to subscribe precisely to the information they are interested in). But the more expressive a pub/sub system is, the more complex the matching algorithm becomes. Thus, the efficiency of the matching algorithm significantly affects both performance and scalability.

2) Fault-tolerance: Depending on the type of the application, there might be the need to ensure different level of reliability. For instance, in a financial system such as the

New York Stock Exchange (NYSE) or the French Air Traffic Control System, reliability is critical [8]. Ensuring a correct dissemination of events despite failures requires a particular form of resiliency. Pub/sub systems which consider event routing based on reliability requirements are rare schemes [9], which proves this challenge has not been sufficiently tackled.

3) Skewed distribution of RDF data: The frequency distribution of terms in RDF data is highly skewed [10], [11]: many triples may share the same predicate (e.g., *rdf:type*). This distribution prevents scalability from algorithms that base their partitioning on these values.

In addition to the challenges which have been previously introduced, some orthogonal properties related to QoS (Quality of Service), such as data delivery semantics, notifications ordering, security aspects, etc. constitute also major open research challenges that are naturally present in RDF-based pub/sub systems [12].

III. Retrieval model

Nowadays, datasets grow so large that they become awkward to work with. This idea is really well captured by the notion of big data from which knowledge acquisition has to be extracted. To make it feasible, information have to be analyzed and correlated. A solution is CEP engines that recently proposed to leverage information which stem from realtime data, contextual and past information. At high scale, a step towards this direction consists in helping CEP engines to reach their goal by providing both storage and realtime filtering of data of interest in order to minimize the amount of information they have to work with. For that, we propose a retrieval model of the stored data based on both *pull* and *push* mechanisms. The pull mode refers to one-time queries; an application formulates a query to retrieve data which have been already stored. In contrast, the push mode refers to pub/sub and is used to notify applications which register long standing queries and push back a notification each time an event that matches them occurs. Contrary to RACED [13], that also proposes a push mode, the result is not the output of a previous subscription matching but it is aimed for getting past and perennial information.

A. Data model

The data model we introduce hereafter is valid for both the pull and push retrieval modes. It is built on top of quadruples. A quadruple extends the concept of RDF triple by adding a fourth element (usually named context or graph value) to indicate or to identify the data source and the event itself. Indeed, the notion of provenance is essential when integrating data from several sources and more generally to classify data on the web. Finally, each quadruple represents a potential event that may be delivered to a subscriber but also a data that is stored.

However, the number of elements contained by an event (quadruple) is limited. To overcome this drawback, we have introduced the notion of compound event: an event that is made of a non-limited number of quadruples. Supposing that a quadruple is modeled by a 4-tuple $q = (c, s, p, o)$ and a compound event by a set $C = \{q_0, q_1, ..., q_n\}$ then each q of C shares the same context value c in order to allow to identify the quadruples that form this compound event. Moreover, thanks to this abstraction, our content-based pub/sub system can support multi-attribute values, still in compliance with RDF data model.

B. Filter model

Both retrieval modes have their filter model based on SPARQL (SPARQL Protocol and RDF Query Language) [14], another W3C specification that is usually used to retrieve and manipulate data stored in RDF format with one time queries. This language is suitable to build a very expressive filter model. Even if it could be used as a pull retrieval model, for the push retrieval model some restrictions are required (e.g. we only allow SELECT query form, a pattern applies to one graph value at a time, see below).

SPARQL provides the possibility to formulate a subscription by associating several filter constraints to a quadruple (event), but also to a set of quadruples that belong to the same compound event. This means that several events that are published at different times and that belong to a same compound event may participate to the matching of a subscription by using their common constraints.

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 SELECT ?user ?name ?age WHERE {
3     GRAPH ?g {
4         ?user foaf:name ?name .
5         ?user foaf:age ?age
6     } FILTER (?age >= 18 && ?age <= 25)
7 }
```

Listing 1. Legal SPARQL subscription indicating that only events about users whose age is between 18 and 25 have to be notified.

Listing 1 shows an example of a subscription that is used to deliver a notification each time two events that belong to the same compound event (represented by the graph pattern and its associated variable `?g`) match the constraints. This subscription depicts two types of constraints that may be uttered. The first one is a join constraint. It consists in computing an equi-join condition on the variable `?user` with the events of the same graph that match the triple patterns (a triple that may contain variables for querying unknown values), see lines 3, 4 and 5. The second type of constraints that may be formulated are filter constraints. Filter constraints are shown in the example by using the `FILTER` keyword on line 6. This second type of constraint may contain several logically related predicates.

Here, we introduced joins because unlike in traditional publish/subscribe systems [15] (where the constraints are matched for each event which is asynchronously published) we want to apply the matching on a set of events (compound event), where each event has been published independently. This condition is fundamental because our push retrieval mode is not supposed to act as a CEP engine correlating several compound events. However, our system has to handle two constraints. First, the quadruples that form a compound event are not stored at the exact same time, due to the fact that the system is distributed, i.e. each quadruple can be indexed on any distributed node of the system in an asynchronous way. Second, a join constraint is limited to a set of quadruples that belong to the same compound event.

IV. Addressing These Challenges

Our aim is to provide an Internet wide system that fulfill challenges introduced in section II while respecting the model presented in section III. There are already some approaches experimenting how to store and query RDF data using popular cloud technology, but along the pull model mainly. Recently, CumulusRDF [16] proposed to rely upon the Cassandra [17] key-value store, by leveraging its two levels indexing model in order to store RDF triples. The choices they make in CumulusRDF are driven by the need to retrieve RDF data by triple patterns only and not the full expressivity of SPARQL. Their solution requires to build more than one index for each triple to be stored. Even if the lookup performances seem reasonably good, they do not support conjunctive queries (joins), nor simple or complex queries that contain some filter conditions. Also, some solutions combining Map-Reduce and distributed data storage systems (e.g., HDFS, BigTable) [18] have been proposed but they require a configuration phase, and then involve several large data sets movements. Moreover, none of the introduced solutions are well adapted when extreme scalability is expected. Indeed, Peer-to-Peer (P2P) systems have been recognized as a key communication model to build platforms for distributed applications at very large scale [19]. For that reason, our system model is based on the original idea of Content Adressable Network (CAN) [20].

A CAN is a structured P2P network (structured in opposition to unstructured, another category of P2P networks better suited to high churn, which is thus less necessary for private/public cloud environments) based on a d-dimensional Cartesian coordinate space labeled $\mathcal{D}$. This space is dynamically partitioned among all peers in the system such that each node is responsible for indexing and storing data in a *zone* of $\mathcal{D}$ thanks to a standard RDF datastore such as Jena [21]. According to our data model, we use a 4-dimensional CAN in order to associate each RDF term of a quadruple to a dimension of the CAN network. A quadruple to index is a point in a 4-dimensional space.

Distributed pub/sub systems have been extensively studied [6], [15], [22] over the last two decades. Recently, some works such as BlueDove [23] revives this field in a cloud context. However, among these works only a few are concerning pub/sub with RDF data and none are combining storage and pub/sub [24].

Most of the proposed solutions use consistent hashing to map data onto nodes for RDF pub/sub systems or RDF data storage. This means that data have to be indexed several times in order to be retrieved and be handled by the pub/sub matching algorithm. For example, CSBV [25] is a matching algorithm for RDF triples that would imply to index each quadruple 15 times in our case: one indexation for each possible combination without repetition of the RDF terms contained by a quadruple. Our approach which relies on CAN, replaces consistent-hashing by lexicographic ordering in order to use only one index and to support range queries efficiently. In return, subscriptions have to be indexed several times. However, we prefer to trade data duplicates with subscriptions duplicates due to the huge foreseen volume of data.

As for systems which use consistent-hashing, we also have to confront the challenge II-3. But in addition, due to lexicographic order, RDF terms which are lexicographically close may be handled by the same peer and unbalance the load between peers. We propose three solutions based on static and dynamic adaptation to overcome this load balancing issue. The first one simply consists in removing prefixes. Suppose that we have to index two quadruples which differs only by one RDF term. On one hand we have *http://example.org/animal* and on the other hand *http://example.org/jacket*. Also, in the network there are two peers: one managing the range $[a, e)$ and one $[e, j)$. In such a case, if we remove the prefix *http://example.org* the former data will be indexed on the first peer and the latter by the second peer. An additional solution is to have an approach similar to the one proposed in BlueDove. Event sources are aware of the type of data and the range values they will publish. Hence, it is possible to take advantage of this information to preconfigure our P2P network. For example, if we know that RDF data published are about weather in Europe and the value of the key event of the published compound events is between $[-20; 40]$, we can leverage this knowledge to increase the number of peers managing this range of values. Finally, the third solution related to elasticity and thus also tackling challenge II-1 will be to balance the load of peers by using the standard join and leave [1] operations of our P2P system. Indeed, by considering the unpredictable and fluctuating amount of information that may be produced (or removed) by any entity, the system has to be elastic. In contrary to an always-on infrastructure for which the institutions refrain to pay for, the idea is to rely on the notion of Cloud Computing to scale horizontally by adding more nodes (peers) on-demand and to release them whenever possible. But also to scale vertically by offering the possibility to deploy several peers on the machines that are underloaded.

To meet challenge II-1 we expect to develop a matching algorithm that parallelizes and balances as much as possible the matching of compound events. A few algorithms have been proposed to balance the matching but the execution to perform a join between several conjunctions is done sequentially by creating a chain [25]. To improve scalability and performances we also intend to manage burst of new subscriptions and the placement of peers according to geographic information. The former case implies to adapt the number of computing agents in charge of the matching process in each peer. The later, such as proposed in [26], consists in improving latency perceived by Internet wide-users (specially subscribers) as CEP(s) by a

[1] In the future, we wish to offer an RDF data garbage collection operation.

geographical mapping of P2P nodes and proxies[2] on cloud hosts.

Finally, challenge II-2 implies to take into account the replication of RDF data but also the states of the matching algorithm. In the former case, we can replicate the data by using the neighbors of a peer. However, in the latter case, it is less obvious because subscriptions do not have to be lost and duplicate notifications have to be avoided. In our system indexing a subscription (set of related patterns) ends up in duplicating it on several peers. We think about leveraging this behavior to come back into a consistent state for the pub/sub layer in case of failure. In addition we are interested to build our system such that user requested QoS properties may be easily addressed. To make it feasible we introduce configurable proxies lying out of the P2P network which will only store and compute the matching between subscriptions and publications. All messages (requests, responses, notifications, etc.) go through these proxies where they can be handled according to requested QoS properties.

V. Conclusion

In this short paper, we have identified and discussed the challenges which need to be addressed in order to build a scalable cloud based RDF storage offering a pub/sub query service. We currently have a first prototype of our extended version of CAN [27], implemented by using ProActive/GCM technology. ProActive is an asynchronous active-object based middleware offering the notion of asynchronous calls with futures (a promise to get back a response) among distributed objects, extended with the possibility to transparently handle groups of objects and security (e.g., authentication, encryption) for inter-object communications [28]. In addition it provides the notion of multi activity to handle requests concurrently. Also, thanks to ProActive abstraction, any peer or proxy needed to access the CAN network can easily be deployed on any host, be it on a private/public cloud, grid or cluster, desktop machines offering elaborated support to address firewall issues, and more generally issues that may be encountered in such a distributed infrastructure.

Acknowledgment

This work was in part supported by the EU FP7 STREP project PLAY and French ANR project SocEDA.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, no. 5, pp. 28–37, 2001.

[2] O. Lassila and R. Swick, "Resource description framework (rdf) model and syntax," *World Wide Web Consortium, http://www. w3. org/TR/WD-rdf-syntax*, 1999.

[3] S. Decker, S. Melnik, F. Van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks, "The semantic web: The roles of xml and rdf," *Internet Computing, IEEE*, vol. 4, no. 5, pp. 63–73, 2000.

[4] D. Luckham, *The power of events: an introduction to complex event processing in distributed enterprise systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[2]Proxies are representing publishers, subscribers and users of the pull mode.

[5] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic, "Ep-sparql: a unified language for event processing and stream reasoning," in *Proceedings of the 20th international conference on World wide web*. ACM, 2011, pp. 635–644.

[6] A. Carzaniga, D. Rosenblum, and A. Wolf, "Achieving scalability and expressiveness in an internet-scale event notification service," in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*. ACM, 2000, pp. 219–227.

[7] J. Wang, B. Jin, and J. Li, "An ontology-based publish/subscribe system," *Middleware 2004*, pp. 232–253, 2004.

[8] K. Birman, "A review of experiences with reliable multicast," *Software: Practice and Experience*, vol. 29, no. 9, pp. 741–774, 1999.

[9] S. Mahambre and U. Bellur, "Reliable routing of event notifications over p2p overlay routing substrate in event based middleware," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–8.

[10] S. Kotoulas, E. Oren, and F. Van Harmelen, "Mind the data skew: distributed inferencing by speeddating in elastic regions," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 531–540.

[11] A. Harth, J. Umbrich, A. Hogan, and S. Decker, "Yars2: A federated repository for querying graph structured data from the web," *The Semantic Web*, pp. 211–224, 2007.

[12] S. Mahambre, M. Kumar, and U. Bellur, "A taxonomy of qos-aware, adaptive event-dissemination middleware," *Internet Computing, IEEE*, vol. 11, no. 4, pp. 35–44, 2007.

[13] G. Cugola and A. Margara, "Raced: an adaptive middleware for complex event detection," in *Proceedings of the 8th International Workshop on Adaptive and Reflective Middleware*. ACM, 2009, p. 5.

[14] E. Prud'Hommeaux and A. Seaborne, "Sparql query language for rdf," *W3C working draft*, vol. 4, no. January, 2008.

[15] P. Pietzuch and J. Bacon, "Hermes: A distributed event-based middleware architecture," in *Distributed Computing Systems Workshops*. IEEE, 2002, pp. 611–618.

[16] G. Ladwig and A. Harth, "Cumulusrdf: Linked data management on nested key-value stores," in *The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011)*, 2011, p. 30.

[17] A. Lakshman and P. Malik, "Cassandra: A structured storage system on a p2p network," in *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*. ACM, 2009, pp. 47–47.

[18] J. Sun and Q. Jin, "Scalable rdf store based on hbase and mapreduce," in *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, vol. 1. IEEE, 2010, pp. V1–633.

[19] M. Jelasity and A. Kermarrec, "Ordered slicing of very large-scale overlay networks," in *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*. IEEE, 2006, pp. 117–124.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 161–172, 2001.

[21] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *World Wide Web conference*. ACM, 2004, pp. 74–83.

[22] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[23] M. Li, F. Ye, M. Kim, H. Chen, and H. Lei, "A scalable and elastic publish/subscribe service," in *Parallel & Distributed Processing Symposium (IPDPS), IEEE International*. IEEE, 2011, pp. 1254–1265.

[24] I. Filali, F. Bongiovanni, F. Huet, and F. Baude, "A survey of structured p2p systems for rdf data storage and retrieval," *Transactions on Large-Scale Data-and Knowledge-Centered Systems III*, pp. 20–55, 2011.

[25] E. Liarou, S. Idreos, and M. Koubarakis, "Continuous rdf query processing over dhts," in *Proceedings of the 6th international semantic web conference*. Springer-Verlag, 2007, pp. 324–339.

[26] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. USENIX Association, 2010, pp. 2–2.

[27] F. Baude, F. Bongiovanni, L. Pellegrino, and V. Quema. (2011) D2.1 requirements eventcloud. Project Deliverable PLAY. [Online]. Available: http://play-project.eu/documents/viewdownload/3/20

[28] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, and R. Quilici, "Programming, composing, deploying for the grid," *GRID COMPUTING: Software Environments and Tools*, pp. 205–229, 2006.