How to feed Apache HBase with petabytes of RDF data:

An extremely scalable RDF store based on Eclipse RDF4J framework and Apache HBase database

Adam Sotona, and Stefan Negru, MSD IT Prague, Czech Republic
{firstname.lastname}@merck.com

Abstract. Majority of the systems designed to handle big RDF data rely on a single high-end computer dedicated to a certain RDF dataset and do not easily scale out, at the same time several clustered solution were tested and both the features and the benchmark results were unsatisfying. In this paper we describe a system designed to tackle such issues, a system that connects RDF4J[1] and Apache HBase[2] in order to receive an extremely scalable RDF store.

1 Introduction

"The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries"[3]. One of the key aspects of the Semantic Web is RDF[4] (Resource Description Framework), a framework that enables the (re)use, discoverability, accessibility and integration of data from multiple diverse sources. Considering the exponential growth of the Linked Open Data[5] sources, but also the internal information across organizations we can note an increasing need for scalable storage solutions for RDF data. Following an evaluation of existing technologies with high potential of handling big semantic data, we derived a set of requirements for such a system, as follows:

1. Handle petabytes of RDF heterogeneous data and be able to scale linearly as the data grows
2. Share common hardware (ideally a cheap "commodity hardware" cluster)
3. Support SPARQL[6] version 1.1and provide basic rule-based reasoning [3]
4. Provide fast bulk and batch data operations and enable exploration and integration of the datasets
5. Export graphs and selected tuples to down-streaming services and application
6. Support multiple teams and individuals working on various tasks
7. Be open for down-streaming services and applications

[1] Eclipse RDF4J: http://rdf4j.org/
[2] Apache HBase: http://hbase.apache.org/
[3] W3C Semantic Web: https://www.w3.org/2001/sw/
[4] RDF: https://www.w3.org/2001/sw/wiki/RDF
[5] Linked Open Data Cloud: http://lod-cloud.net/
[6] SPARQL 1.1: http://www.w3.org/TR/sparql11-query/

2 Key Components

Two key aspects can be correlated to a scalable Triple/RDF Store and those are a scalable database, and a dependable framework for working with RDF data. Our proposal architecture for such a system is represented in Figure 1.

Eclipse RDF4J [6] represents the framework, selected mostly because of its features and the fact that it facilitates the achievement of the identified (including Handle petabytes of data and the support of SPARQL 1.1). Other reasons consist in its extensibility through Storage And Inference Layer (SAIL) to support various RDF stores and inference engines, its test coverage and the extensive documentation.

Apache HBase is the Hadoop[7] database, a distributed and scalable big data store. It is designed to scale up from single servers to thousands of machines, thus providing a flexible and reliable means of handling big RDF data.

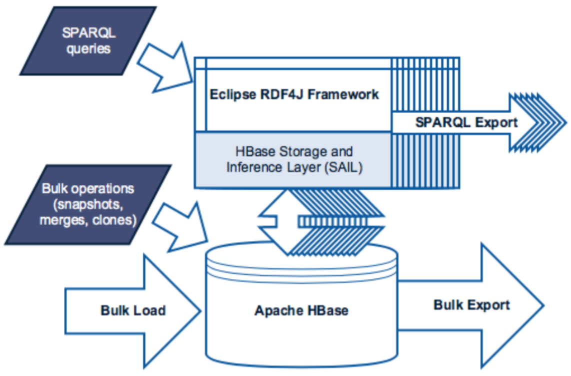
SPARQL queries
Eclipse RDF4J Framework
SPARQL Export
HBase Storage and Inference Layer (SAIL)
Bulk operations (snapshots, merges, clones)
Bulk Load
Apache HBase
Bulk Export

Fig. 1. HBase RDF Store Diagram

Bulk Data Operations

Bulk data operations are critical to the overall system efficiency. A system capable to hold and query hundreds of billions of triples must be also able to load them in, create snapshots, clone and export in bulk operations.

Our current bulk load is implemented as a simple MapReduce application consuming RDF data in any standard format and producing HBase table region files directly by each reducer task. Performance of the bulk load scales linearly with the Hadoop cluster. Measurement of load and indexing performance shows in average about 40.000 quads per second per each cluster node. Bulk Export is also a MapReduce application, by exporting the whole dataset in parallel - measured performance shows in average about 400.000 quads per second per each cluster node. Bulk merge is multi-step operation composed from snapshotting of the source datasets and direct loading of all HBase region files into one merged HBase table.

[7] Apache Hadoop. (n.d.). https://hadoop.apache.org/

In order to achieve such performance, a must is a proper RDF mapping to Apache HBase, as it provides much needed performance and scalability of the whole system. After several iterations of trying various mapping approaches we have found the optimal RDF mapping with satisfying performance, a mapping represented in Table 1.

RDF Triple: <SUBJ1> <PRED1> <OBJ1>

RDF Quad: <SUBJ2> <PRED2> <OBJ2> <CTX2>

<= Ordered by Row Keys	Apache HBase Table (single Column Family)			
	Regions	Row Keys	Column Qualifiers	Values are not used (empty)
	SPO	0<SUBJ1 hash><PRED1 hash><OBJ1 hash>	<SUBJ1><PRED1><OBJ1>	
		0<SUBJ2 hash><PRED2 hash><OBJ2 hash>	<SUBJ2><PRED2><OBJ2><CTX2>	
	POS	1<PRED1 hash><OBJ1 hash><SUBJ1 hash>	<SUBJ1><PRED1><OBJ1>	
		1<PRED2 hash><OBJ2 hash><SUBJ2 hash>	<SUBJ2><PRED2><OBJ2><CTX2>	
	OSP	2<OBJ1 hash><SUBJ1 hash><PRED1 hash>	<SUBJ1><PRED1><OBJ1>	
		2<OBJ2 hash><SUBJ2 hash><PRED2 hash>	<SUBJ2><PRED2><OBJ2><CTX2>	
	CSPO	3<CTX2 hash><SUBJ2 hash><PRED2 hash><OBJ2 hash>	<SUBJ2><PRED2><OBJ2><CTX2>	
	CPOS	4<CTX2 hash><PRED2 hash><OBJ2 hash><SUBJ hash>	<SUBJ2><PRED2><OBJ2><CTX2>	
	COSP	5<CTX2 hash><OBJ2 hash><SUBJ2 hash><PRED2 hash>	<SUBJ2><PRED2><OBJ2><CTX2>	

Table. 1. RDF Mapping to Apache HBase

RDF4J SAIL Implementation

Storage and Inference Layer (SAIL) is a key interface of the RDF4J framework to communicate with various storage and/or inference sub-systems. A minimalistic implementation of the HBase RDF4J SAIL works perfectly even it has not been originally designed for big data operations. We optimized the performance also by stripping the transactions support and streamlining the dataflow through the RDF4J SAIL and through the RDF4J framework. Brief performance measurement has been performed using Berlin SPARQL Benchmark [4] and its Explore Use Cases query mix. For 10 billion triples dataset the average QMPH is 7,8 per single cluster node. With excluded query #5 from the query mix it shows more promising 588 QMPH per single cluster node involved in the benchmark.

With regard to reasoning, our target datasets size limits us to basic interferences over RDFS+ [1] and similar set of rules. Based on batch-processing mode and high storage limits in HBase, we have decided to use batch processes to physically materialize inferred triples. More precisely we developed a tool performing the reasoning based on SPARQL in a batch mode.

Parallel SPARQL Evaluation Strategies

RDF4J framework comes with a standard SPARQL 1.1 compliant evaluation strategy. This strategy uses single-threaded pull model where the data are requested from the underlying RDF store when needed in the evaluation model. This model shows excelling results on all low-latency RDF stores, however it is not performing enough for distributed storage systems. Our improved SPARQL evaluation strategy uses parallel push model. In this model each SPARQL query tree is expanded, requests are processed asynchronously and data are pushed back for evaluation when ready. A proper prioritization of the asynchronous processes is a key to let the data flow through the system with minimal latency and without memory overflows.

Another experimental parallelization strategy is targeted for parallel SPARQL-based export, which makes uses multiple distinct jobs working on the same SPARQL query and custom SPARQL filtering function to avoid duplicate work and duplicate data. Parallel architecture allows us also to spawn multiple SPARQL endpoints and balance the load between them. As a result of the parallel architecture the above-mentioned export and query performance measurements can be multiplied by the number of involved cluster nodes up to the cluster size.

3 Related work

Similar directions of research have been used by Jianling Sun et.al. [5] in storing RDF in HBase related mapping presented in Table 1 and Apache RYA [2] has been developed in the same spirit. Nevertheless Apache RYA [3] is still an Apache Incubator as project, and current requirements are divergent, yet there is a vision that our efforts might merge in the future.

4 Conclusion and Future Work

In this paper we described the process of developing a scalable RDF system by focusing on satisfying what we deemed as the minimal set of requirements for such as system, some worth mentioning are: *Storage capacity in tens of terabytes of RDF data and potentially up to petabytes; Support for thousands of distinct datasets, in a single HBase table each; Down-streaming services and applications are supported through SPARQL, HBase and Java APIs.* All things considered we are seeking to answer more questions by the implementation of such as system and look into integrating other improvements that need further development and testing.

References

1. Allemang, D., & Hendler, J. A. (2011). Semantic Web for the working ontologist: Effective modeling in RDFS and OWL. Waltham, MA: Morgan Kaufmann/Elsevier.
2. Punnoose, R., Crainiceanu, A., & Rapp, D. (2012). Rya. Proceedings of the 1st International Workshop on Cloud Intelligence - Cloud-I '12.
3. Apache Rya (incubating). (n.d.). http://rya.incubator.apache.org/
4. Berlin SPARQL Benchmark (BSBM). (n.d.). http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/
5. Jianling Sun and Qiang Jin, "Scalable RDF store based on HBase and MapReduce," *(ICACTE)*, Chengdu, 2010, pp. V1-633-V1-636.