Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store

Vaibhav Khadilkar[1], Murat Kantarcioglu[1], Bhavani Thuraisingham[1], and Paolo Castagna[2]

[1] The University of Texas at Dallas
[2] Talis Systems Ltd.

Abstract. Lack of scalability is one of the most significant problems faced by single machine RDF data stores. The advent of Cloud Computing has paved a way for a distributed ecosystem of RDF triple stores that can potentially allow up to a planet scale storage along with distributed query processing capabilities. Towards this end, we present Jena-HBase, a HBase backed triple store that can be used with the Jena framework. Jena-HBase provides end-users with a scalable storage and querying solution that supports all features from the RDF specification.

1 Introduction

The simplest way to store RDF triples comprises a relation/table of three columns, one each for *subjects*, *predicates* and *objects*. However, this approach suffers from lack of scalability and abridged query performance, as the single table becomes long and narrow when the number of RDF triples increases [1]. The approach is not scalable since the table is usually located on a single machine. In addition, query performance is diminished since a query requires several self-joins with the same table. There have been several approaches, for *e.g.* [2], which modify the single-table storage schema to improve query performance. Nevertheless, these approaches suffer from the scalability problem, which can be solved by moving from a single-machine to a multi-machine configuration. The cloud computing paradigm has made it possible to harness the processing power of multiple machines in parallel. Tools such as Hadoop and HBase provide advantages such as fault tolerance and optimizations for real time queries. In this paper, we present Jena-HBase[3], a HBase backed triple store that can be used with the Jena framework along with a preliminary experimental evaluation of our prototype.

Our work focuses on the creation of a distributed RDF storage framework, thereby mitigating the scalability issue that exists with single-machine systems. The motivation to opt for Jena is its widespread acceptance, and its built-in support for manipulating RDF data as well as developing ontologies. Further, HBase was selected for the storage layer for two reasons: (i) HBase is a column-oriented store and in general, a column-oriented store performs better than row-oriented stores [1]. (ii) Hadoop comprises Hadoop Distributed File System (HDFS), a distributed file system that stores data, and MapReduce, a framework for processing data stored in HDFS. HBase uses HDFS for data storage but does not require MapReduce for accessing data. Thus, Jena-HBase does not require the implementation of a MapReduce-based query engine for executing queries on RDF

[3] https://github.com/castagna/hbase-rdf, https://github.com/vaibhavkhadilkar/hbase-rdf

triples. In contrast, existing systems that use a MapReduce-based query engine for processing RDF data are optimized for query performance, however, they are currently unable to support all features from the RDF specification. Our motivation with Jena-HBase is to provide end-users with a cloud-based RDF storage and querying API that supports all features from the RDF specification.

Our contributions: Jena-HBase provides the following: (a) A variety of custom-built RDF data storage layouts for HBase that provide a tradeoff in terms of query performance/storage. (b) Support for reification, inference and SPARQL processing through the implementation of appropriate Jena interfaces.

2 Jena-HBase Architecture

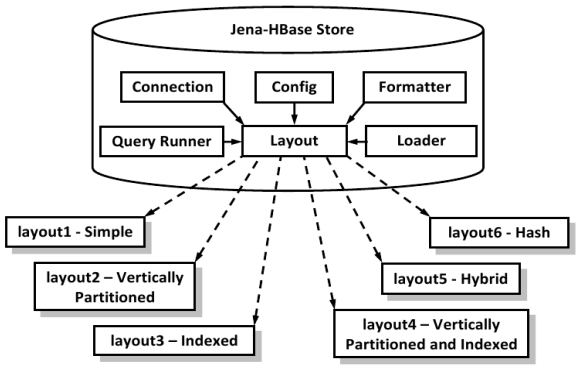

Fig. 1. Jena-HBase - Architectural Overview

Fig. 1 presents an overview of the architecture employed by Jena-HBase. Jena-HBase uses the concept of a store to provide data manipulation capabilities on underlying HBase tables. A store represents a single RDF dataset and can be composed of several RDF graphs, each with its own storage layout. A layout uses several HBase tables with different schemas to store RDF triples; each layout provides a tradeoff in terms of query performance/storage. All operations on a RDF graph are implicitly converted into operations on the underlying layout. These operations include: (a) Formatting a layout, *i.e.*, deleting all triples while preserving tables (**Formatter** block). (b) Loading-unloading triples into a layout (**Loader** block). (c) Querying a layout for triples that match a given $< S, P, O >$ pattern (**Query Runner** block). (d) Additional operations include the following: (i) Maintaining an HBase connection (**Connection** block). (ii) Maintaining configuration information for each RDF graph (**Config** block).

We briefly give a summary of the storage schema used by each layout in Table 1. A detailed description of each layout is further given in [3].

3 Experimental Evaluation

We have performed benchmark experiments using SP^2Bench (non-inference queries) [4] and LUBM (inference queries) [5] to determine the best layout currently available in Jena-HBase, as well as to compare the performance of the best layout with Jena TDB. We have compared Jena-HBase only with Jena TDB and not with other Hadoop-based systems for the following reasons: (i) Jena TDB gives

Table 1. Storage schemas for Jena-HBase layouts

Layout Type	Storage Schema
Simple	3 tables each indexed by subjects, predicates and objects
Vertically Partitioned (VP)	For every unique *predicate*, two tables, each indexed by subjects and objects
Indexed	Six tables representing the six possible combinations of a triple namely, SPO, SOP, PSO, POS, OSP and OPS
VP and Indexed	VP layout with additional tables for SPO, OSP and OS
Hybrid	Simple + VP layouts
Hash	Hybrid layout with hash values for nodes and a separate table containing hash-to-node mappings

the best query performance of all available Jena storage subsystems. (ii) The available Hadoop-based systems do not implement all features from the RDF specification. In this section, we show results only for Q1 and Q9 of SP^{2}Bench and Q1 and Q10 of LUBM, however, these results are indicative of the overall trend [3]. Additionally, the figures only show query times and do not include loading times. Finally, as a part of the procedure to determine the best layout, we ran both benchmarks over several graph sizes, but we show results only for a graph of 250137 triples for SP^{2}Bench and for a graph of 5 universities ($\approx$ 560K triples) for LUBM (Fig. 2). Although we used a small graph size, it is still sufficient for determining the best Jena-HBase layout. Since LUBM contains inference queries, we used the Pellet reasoner (v2.3.0) to perform inference.

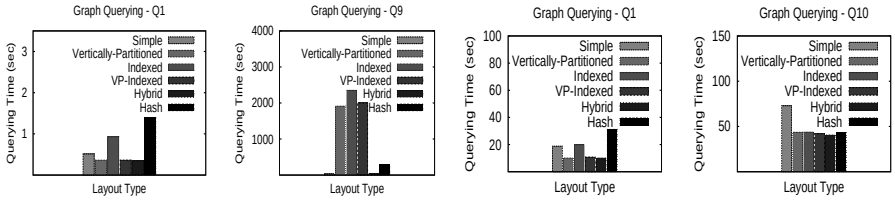

Fig. 2. Comparison of layouts for SP^{2}Bench (Q1 and Q9) and LUBM (Q1 and Q10)

Fig. 2 shows a comparison of all layouts for Q1 and Q9 of SP^{2}Bench and Q1 and Q10 of LUBM. We see that the Hybrid layout gives the best results since it combines the advantages of the Simple (Q9 of SP^{2}Bench and Q10 of LUBM) and VP (Q1 of SP^{2}Bench) layouts. The Indexed, VP-Indexed and Hash layouts require longer querying times, since they require multiple row lookups in the SPO, SOP, PSO, POS, OSP and OPS tables (Indexed case) or the SPO, OSP and OS tables (VP-Indexed case) or the mapping table (Hash case).

Fig. 3 shows a comparison of the Hybrid layout with Jena TDB for increasing graph sizes. Note that Jena-HBase values have been scaled down for Q1 (by 1000) and Q9 (by 100) of SP^{2}Bench and for Q1 (by 10) of LUBM for a clear comparison. We see that TDB outperforms the Hybrid layout in the 1M to 25M range for SP^{2}Bench and in the $N = 50$ to 500 range (N is the number of universities) for

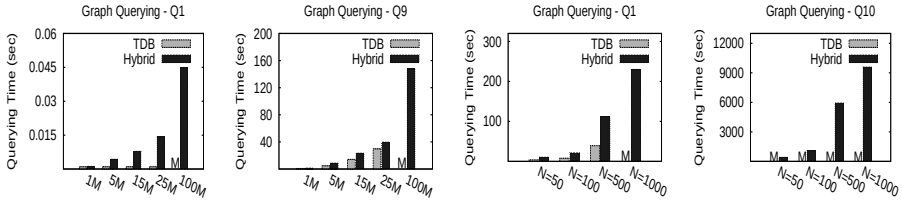

Fig. 3. Comparison of Jena TDB *vs.* Jena-HBase Hybrid layout for SP^{2}Bench (Q1 and Q9) and LUBM (Q1 and Q10). Note that **M** denotes an **Out Of Memory** exception.

Q1 of LUBM. This is expected since for these ranges TDB is able to create and maintain the necessary B+ graph indices in memory, thus resulting in a shorter query execution time. Jena-HBase requires multiple graph pattern matches on increasing graph sizes over a distributed cluster, thus making it slower than TDB. We also observe that TDB fails to execute Q10 of LUBM for the $N = 50$ to 500 range, since the test program runs out of memory during the process of inference for this range. The Hybrid layout successfully executes Q10 for this range, since the reasoner is able to construct the necessary inference related data structures. Finally, we observe that Jena-HBase is more scalable than TDB which fails to construct graphs with 100M triples for SP^{2}Bench and $N = 1000$ universities for LUBM, thereby preventing the execution of any query on these graphs.

4 Conclusion

In this paper, we show that creating a distributed RDF storage framework with existing cloud computing tools results in a scalable data processing solution. Additionally, our solution maintains a reasonable query execution time overhead when compared with a single-machine RDF storage framework (*viz.* Jena TDB).

5 Acknowledgements

This work was partially supported by The Air Force Office of Scientific Research MURI-Grant FA-9550-08-1-0265 and Grant FA-9550-08-1-0260, National Institutes of Health Grant 1R01LM009989, National Science Foundation (NSF) Grant Career-CNS-0845803, and NSF Grants CNS-0964350, CNS-1016343 and CNS-1111529. We thank Dr. Robert Herklotz for his support.

References

1. D. J. Abadi, A. Marcus, S. Madden, and K. J. Hollenbach. Scalable Semantic Web Data Management Using Vertical Partitioning. In *VLDB*, pages 411–422, 2007.
2. K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF Storage and Retrieval in Jena2. Technical report, HP Laboratories, 2003.
3. V. Khadilkar, M. Kantarcioglu, P. Castagna, and B. Thuraisingham. Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store. Technical report, 2012. `http://www.utdallas.edu/~vvk072000/Research/Jena-HBase-Ext/tech-report.pdf`.
4. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP^{2}Bench: A SPARQL Performance Benchmark. In *ICDE*, pages 222–233, 2009.
5. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.