MITIGATION OF LARGE-SCALE RDF DATA LOADING WITH THE EMPLOYMENT OF A CLOUD COMPUTING SERVICE

Hyun Namgoong[1], Harshit Kumar[1,2] and Hong-Gee Kim[1]
[1]*Biomedical Knowledge Engineering Lab, Seoul National University, Seoul, Korea*
[2]*Department of Computer Science, University of Suwon, Hwaseong, Korea*

Keywords: RDF data store, Data warehouse, Cloud computing service, Sesame.

Abstract: An expanding need for interoperability and structuralization of web data has made use of RDF (Resource Description Framework) plentiful. To guarantee a common usage of the data within various applications, several RDF stores providing data management services have been developed. Here, we represent a systematic approach to solve a late latency problem of data loading of the stores. It enables a fast loading performance for very large size of RDF data, and it is proven with an existing RDF store. This approach employs a cloud computing service and delegates preparation works to the machines which are temporarily borrowed at little payment. Our implementation for a native version of the Sesame RDF Repository was tested on LUBM 1000 University data (138 million triples), and it showed a local store loading time of 16.2 minutes with additional preparation time on a cloud service taking approximately an hour, which can be reduced by adding supplemental machines to the cluster.

1 INTRODUCTION

As a key technology in representing interlinking data with heterogeneity, RDF (Resource Description Framework) is becoming a common way to publish structured data throughout the web (Bizer, 2007). Although there are difficulty and inefficiency in RDF data processes, the efforts for expansion of uses of the data expression formats are still evolving. The possibility of interoperability and inference are competitive edges of RDF data.

Providing an interactive querying service on RDF data has been a huge research issue in the Semantic Web research area. Several RDF stores have been developed during past years (Erling, 2007)(Broekstra, 2002)(Liu 2005). The stores evaluating a SPARQL query by retrieving stored triples recently met a scalability issue with an expansion of RDF data.

The point of this paper is focused on bulk data loading in the RDF data stores. Loading a large amount of data is an essential task that stores usually meet at the initial time. Preparation works like assigning unique IDs for each unique value and building index systems, e.g., B tree and Bitmap precedes in the stores to boost data access speed during querying. Those works are time consuming process and it is hard to coordinate with multiple machines. Therefore, a scaled loading capability within a reasonable time is one of the issues for the improvement of RDF store, because, their poor performance obstructs the use of web-scale knowledge bases such as conspicuously evolving linked data (Bizer, 2007).

This paper introduces a systematic approach employing a cloud computing service which takes a local store's preparation works for mitigating bulk data loading. This paper depicts how the loading preparation works can be translated to a series of batch processes to be handled in a MapReduce cluster which can be easily usable with computing resources of cloud computing service.

2 RELATED WORKS

There are several RDF stores developed by semantic web research groups being widely employed for handling RDF data. Representative systems like Sesame, Virtuoso, and Jena provide RDF data management services with their own data storing schema. The performances of a repository are measurable with test data sets and sample queries of widely known Semantic web repository benchmarks

Namgoong H., Kumar H. and Kim H..
MITIGATION OF LARGE-SCALE RDF DATA LOADING WITH THE EMPLOYMENT OF A CLOUD COMPUTING SERVICE.
DOI: 10.5220/0003142204890492
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2010), pages 489-492
ISBN: 978-989-8425-29-4

like LUBM (Guo, 2005) and SP2B (Schmidt, 2008). Current benchmarks show a many number of queries can be dealt with in nearly real time with some heuristic methods like query optimizations (Schmidt, 2008). However, in a point of data loading, the stores are still suffering from insufficient performance which is a great bottleneck in use of RDF stores. It is involved with even several gigabytes data. and, it is getting worse in a linier scale with increasing size of data

Cloud Computing Services are a recent trend which introduced the concept of the borrowing of computing resources from the web. Web based service sites like Amazon and Google opened their idle resources to public users. A service user can borrow some of their computing resources to process a large-scale data that recalls a massive amount of computation.

By adapting a cloud computing service for helping a part of data loading process, the proposed system makes the RDF stores relieved from a burden of the data loading. The borrowed machines in the cloud computing service handle the preparation works, and they hand over the pre-processed RDF data to the RDF stores, so that, the store can load it easily. This paper explains an implementation of those preparation works using MapReduce framework which provides black box interface for archiving a batch data processing work with the multiple machines network-wired.

MapReduce (Dean, 2008) is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function which merges all intermediate values associated with the same intermediate key. The computation takes a set of input key/value pairs, and produces a set of output key/value pairs.

3 SACE

We implemented 'with Cloud Version' of Sesame Native Store named SACE (Sesame Sail with a Cloud Computing Environment). The implementation consists of a realization of multi-machine works of a cluster constructed on the Amazon Web Service and a simple manipulation on Sesame as a local store.

The multi-machine work is designed as a serial execution of five steps. An execution of such processes prepares four internal data files for a Sesame local store, three value encoding related files explained above plus a namespace file. Sorted triple files corresponding to the sequences of triple indexes are also provided after the final step.

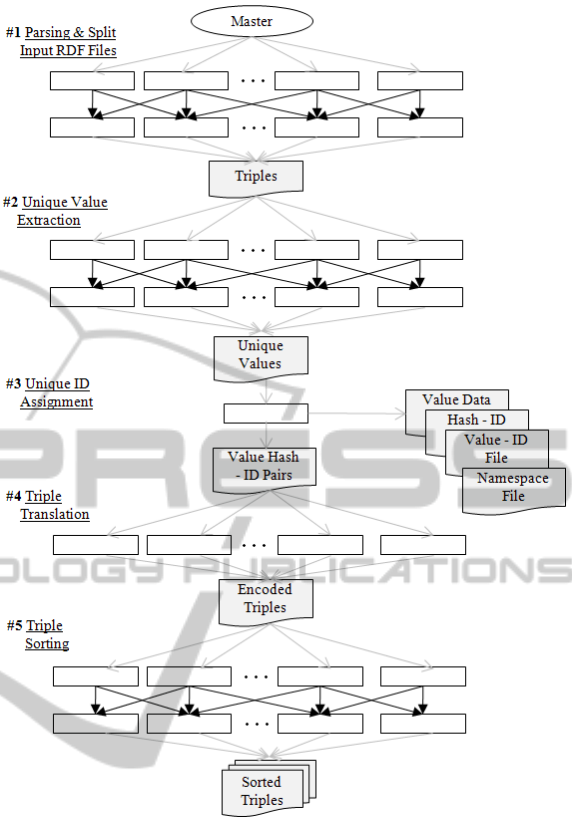

Figure 1: Designed Processed on a Cloud Computing.

Figure 1 describes those five steps for the preparation helping Sesame's loading process. Each step is executed by the rented machines from a cloud computing service that is depicted as squares. The whole processes are done with three MapReduce (M-R) works, single machine work and a multi-peer work.

- Step #1. Parsing & Splitting Input RDF Files

RDF Parsing does not play a huge part from the viewpoint of time consumption. Less than 1 percent from the whole overload occupied by the job as described previously. Although, this work does help the loading process of a local server, it is still required to perform the other jobs in a cloud service. This work is designed as a type of M-R programming. The mappers charging the distribution of data to the reducers are equipped with a Jena RDF parser. Each mapper parses a series of input files shared by a master peer and splits them into triples in parallel to be sent to the reducers. A random integer is made to evenly distribute those triples to

multiple numbers of reducers. The reducers deliver a single file including whole triples. The namespaces of each are specially gathered using a specified key value to be sent to a specific reducer.

- Step #2. Unique Value Identification

At Step 2 all values occurring in a whole triple from Step 1 are extracted as unique values. Execution of this step can be archived with a characteristic of M-R. When a value is assigned as a key for the M-R process, the value is sent to a reducer which only deals with the same values as that value. The only code executed in reducers is omitting its key once disregard values. A list of unique values is the output of this step.

- Step #3. Unique ID Assignment

Step 3 is need for the preservation of the ID of the decoding files. This process is designed to be handled by a single computing machine so as not to suffer the problems of multiple peer synchronization during the assignment of an ID for each value. Three decoding files, the value data, hash-id, value-id files and the namespace file are provided with unique values from the previous step during this step. The unique value hash-id is also made, so that, triples can be translated into a set of IDs. Hash code based on segmentation of these data is required for quick access to them.

- Step #4. Triple Translation

All triples should be translated into ID based triples to be provided as a sorted triple set corresponding to triples index options. To archive translation work in the multiple peers in parallel, the value hash-id file is delivered to the machine involved in this step. Using the hash-id pair file, every triple is rewritten as a set of IDs.

- Step #5. Triple sorting

The final step is the sorting that arranges ID based triples in sequential order. This step works similarly to the second step. The different selections of a key value were added to make diversely sorted triples corresponding to the triple index sequences.

4 EXPERIMENTAL RESULT

This section describes two experimental results 1) expected times for the execution of designed M-R processes. 2) loading time in a local store with pre-processed RDF data.

There is a trade-off between cost and performance, in M-R processes. Because, the time spents of these processes are also dependent on the number of machines employed from the cloud computing service. We try to describe a brief benchmark result that includes the overall tendencies rather than a definitive benchmarking of the processes which is not much critical issues. Therefore, some unrespectable and short times for operation transfers, as well as machine configuration and data transmission times are omitted, and the time spent for each step is described.

The elapsed times for each step are depicted in Figure 2. The graph shows the result of step 1, step 2, step 4, and step 4, respectively. It is tested for the following numbers of nodes, 5, 10, 20, 40, and 80. With 80 nodes of computing machines for LUBM 1000 univ data, step 1 takes about 35 minutes, 2.5 minutes for step 2, a minute for step 4, and 9 minutes for step 5. Step 4 handled by a single machine takes about 17 minutes for the data. The process of step 4, sorting encoded triples, requires several executions for each distinct order, so, this step takes relatively much more time than step 2.

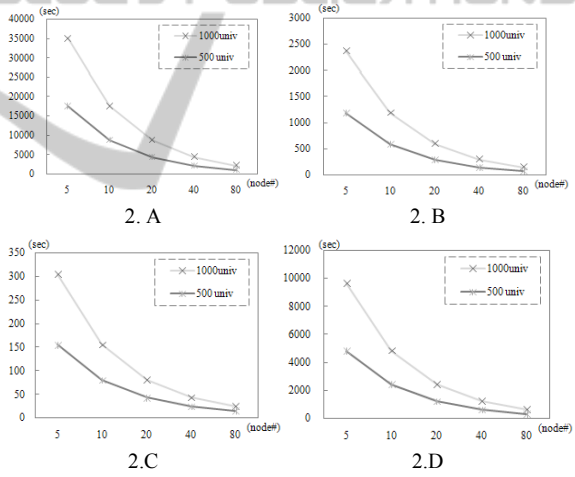

Figure 2: Benchmark Times for Steps 1, 2, 4, and 5.

Also, a payment for the use of the cloud computing service is a criterion to judge the practical usefulness of this approach. For each size of data (500 univ, 1000 univ), about 8$ and 16$ are charged, respectively. The payment was calculated with the multiplication of a standard cost for a medium size of High-CPU On-Demand Instances ($0.20 per hour and a number of marked machines plus one for a master. An employment of many machines shows a sustained scale by the enlargement of the number of nodes because of decreasing whole data process time.

As described, after the process at the cloud computing service, the rests of loading process need

be archived at the local store. In the provided implementation with Sesame, the construction of the B tree needs to be archived at the local store. As we described, ordered triples data will help the construction task fast.

To get the experimental results of the local RDF store's work, we made a revision on the Native version of Sesame. The revised Sesame reads value storing files handed over by a master node of the M/R cluster. Then, it executes pre-processing on behalf of the store. The environment is as follows: Ubuntu 2.24.1 of Linux, CPU: Intel quad core 2.8GH, Memory: 3.5GB Samsung, HDD: 500GB S-ATAII (7200 rpm) with single partition, Java SDK 1.6 version, JVM Heap Size: Minimum 1024 MB, Maximum 2048 MB.

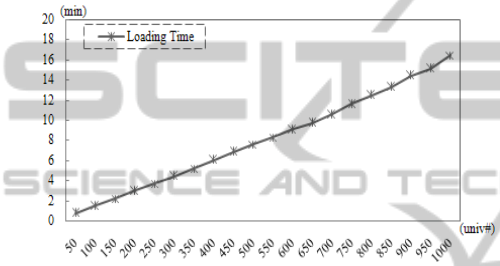

Figure 3: Benchmark Times in Local Store.

The result in the Figure 3 shows very fast loading time on the data set from LUBM 50 univ ~ 1000 univ. It only takes about 16 minutes for 138 million triples (1000 univ) with higher TPS rates.

Also, for extremely large sized data sets which include non realistic numbers of triples, it also shows a reasonable loading time. A passable process time on a cloud computing service for those sizes of data could be archived similarly with the computing times we have shown.

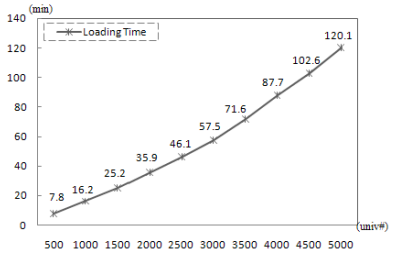

Figure 4: Very Large-Scale Benchmarks in Local Store.

5 CONCLUSIONS

We presented the practical approaches to dynamically reduce large-scale RDF data loading with the aid of a cloud computing service. Experimental results giving insight on the overall time spent are also provided a conversion from a single machine-based job to multiple machine work. For such conversions, M-R programming and simple parallel processing are embodied. The implementation for a native version of Sesame RDF Repository delivers a very fast loading time marked a local store loading time of 16.2 minutes with additional preparation time on a cloud service, which can be lessened by adding supplemental machines.

ACKNOWLEDGEMENTS

This work was supported in part by MKE & KEIT through the Development of Independent Component based Service-Oriented Peta-Scale Computing Platform Project.

REFERENCES

Bizer, C., Cyganiak, R., Heath, T., 2008. How to Publish Linked Data on the Web, Available at: *http://www4.wiwiss.fu-berlin.de/bizer/pub/ LinkedData Tutorial/20070727/.*

Broekstra, J., Kampman, A., Harmelen., F., 2002. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, *International Semantic Web Conference (ISWC 2002).*

Guo, Y., Pan, Z., Heflin., J., 2005. LUBM: A Benchmark for OWL Knowledge Base Systems, *Journal of Web Semantics3*.

Schmidt, M., Hornung, T., Küchlin, N., Lausen, G., Pinkel, C., 2008. An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario, *International Semantic Web Conference (ISWC 2008).*

Schmidt, M., Hornung , T., Küchlin, N., Lausen, G., Pinkel, C., 2008. An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario, *International Semantic Web Conference (ISWC 2008).*

Liu, B., Hu, B., 2005. An evaluation of RDF storage systems for large data applications, *In Proceedings of the First International Conference on Semantics, Knowledge and Grid*.

Erling, O., and Mikhailov, I., Towards Web-Scale RDF, *Available at: http://virtuoso.openlinksw.com/ dataspace/dav/ wiki/Main/VOSArticleWebScaleRDF*.

Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters, *Communications of the ACM*, v.51.