Journal of Database Management
Volume 30 • Issue 1 • January-March 2019

Map-Side Join Processing of SPARQL Queries Based on Abstract RDF Data Filtering

Minjae Song, Yonsei University, Seoul, South Korea
Hyunsuk Oh, Yonsei University, Seoul, South Korea
Seungmin Seo, Yonsei University, Seoul, South Korea
Kyong-Ho Lee, Yonsei University, Seoul, South Korea

ABSTRACT

The amount of RDF data being published on the Web is increasing at a massive rate. MapReduce-based distributed frameworks have become the general trend in processing SPARQL queries against RDF data. Currently, query processing systems that use MapReduce have not been able to keep up with the increase of semantic annotated data, resulting in non-interactive SPARQL query processing. The principal reason is that intermediate query results from join operations in a MapReduce framework are so massive that they consume all available network bandwidth. In this article, the authors present an efficient SPARQL processing system that uses MapReduce and HBase. The system runs a job optimized query plan using their proposed abstract RDF data to decrease the number of jobs and also decrease the amount of input data. The authors also present an efficient algorithm of using Map-side joins while also using the abstract RDF data to filter out unneeded RDF data. Experimental results show that the proposed approach demonstrates better performance when processing queries with a large amount of input data than those found in previous works.

KEYWORDS

Abstract RDF Data, HBase, MapReduce, Map-Side Join, SPARQL Query Processing

1. INTRODUCTION

With the dissemination of the Resource Description Framework (RDF) and the SPARQL query language, the number of organizations that use RDF to publish data on the Web is growing, and the total amount of RDF data that have been published is also increasing at a staggering rate. RDF data and SPARQL query have been used in a wide range of tasks, such as semantic stream processing (Barbieri et al., 2009; Chun et al., 2017), spatiotemporal query processing (Hu et al., 2015; Jaziri et al., 2015; Eom et al., 2017) and analyzing ontological models (Rivero et al., 2015), etc.

Processing SPARQL queries against a large volume of RDF data is a challenging task. Most of the conventional methods center on developing scalable RDF query engines. RDF stores like

DOI: 10.4018/JDM.2019010102

Jena (Carroll et al., 2004), RDF-3X (Neumann et al., 2010), 3store (Harris et al., 2003), Hexastore (Weiss et al., 2008), SW-Store (Abadi et al., 2009) and Sesame (Broekstra et al., 2002) use a centralized approach. As the amount of RDF data increases, it is becoming harder to store and process them on a single machine. There is also a distributed approach to storing RDF data in a distributed relational database system, in which SPARQL queries can be converted to SQL versions (Husain et al., 2011). RDF stores like SHARD (Rohloff et al., 2010) and HadoopRDF (Husain et al., 2011) use a distributed computing system to store RDF data across numerous machines (Huang et al., 2011; Picalausaal et al., 2012). Most of these systems use Hadoop to execute joins between subsets of RDF data. The shuffling stage and large network usage between the Map and Reduce stages of a MapReduce framework result in performance decrease in the conventional Reduce-side joins. To overcome the performance degradation, MAPSIN (Schätzleet et al., 2012) and RDFChain (Choi et al., 2013) introduce Map-side joins to SPARQL query processing. There still exists the issue of large network usage when several jobs are needed to execute a query. So, TriAD (Gurajada et al., 2014) adopts a join-ahead pruning via graph summarization to prune triples ahead of join operations.

Overall, the following issues exist with the conventional methods:

- The existence of shuffling phase and network overload between the Map and Reduce phases of a MapReduce framework. Shuffling causes traffic due to a large amount of intermediate results;
- The performance issue of saving intermediate results to a disk. When moving from a Map phase to a Reduce phase, the intermediate results should be saved to a disk before the Reduce phase, resulting in a number of disk I/Os. The disk I/O is a major overhead that should be avoided;
- The increasing number of jobs to execute a query. When executing a complex or long running query, multiple jobs may be needed for the query to be carried out successfully. When multiple jobs are connected sequentially, both the first and second issues outlined above may occur.

We propose an efficient Map-side join-processing approach that uses MapReduce and HBase to run a job optimized query plan, which uses our proposed abstract RDF data to filter intermediate results. The main contributions of the proposed method are as follows:

- Design an efficient HBase schema, which holds all the data for a specific join variable in a single row. The schema allows for multiple triple patterns with the same join variables to be evaluated in one job. Partition information is encoded into the schema to load adjacent RDF data;
- Reduce the number of jobs in a MapReduce framework by using a job optimization algorithm;
- Use partition information to create a filter and use it to reduce the input size of a map job;
- Demonstrate that the proposed method performs better when processing queries with a large amount of input, compared with the conventional method.

The paper is organized as follows. Section 2 gives a basic introduction to the background knowledge and Section 3 presents a discussion of related work. Section 4 and 5 show the proposed architecture of our system and explain the RDF data loading and query processing. Section 6 presents comparisons with the conventional method. Section 7 summarizes conclusions and makes plans for future work.

2. BACKGROUND

This section introduces an overview of the key technologies related with the proposed approach.