SPIDER : A System for Scalable, Parallel / Distributed Evaluation of large-scale RDF Data

Hyunsik Choi, Jihoon Son, YongHyun Cho, Min Kyoun Sung, Youn Dohn Chung
Department of Computer Science and Engineering
College of Information and Communication,
Korea University, Seoul, Republic of Korea
{hyunsikchoi, jihoonsoon, yonghyun, mj999, ydchung}@korea.ac.kr

ABSTRACT

RDF is a data model for representing labeled directed graphs, and it is used as an important building block of semantic web. Due to its flexibility and applicability, RDF has been used in applications, such as semantic web, bioinformatics, and social networks. In these applications, large-scale graph datasets are very common. However, existing techniques are not effectively managing them. In this paper, we present a scalable, efficient query processing system for RDF data, named SPIDER, based on the well-known parallel/distributed computing framework, Hadoop. SPIDER consists of two major modules (1) the graph data loader, (2) the graph query processor. The loader analyzes and dissects the RDF data and places parts of data over multiple servers. The query processor parses the user query and distributes sub queries to cluster nodes. Also, the results of sub queries from multiple servers are gathered (and refined if necessary) and delivered to the user. Both modules utilize the MapReduce framework of Hadoop. In addition, our system supports some features of SPARQL query language. This prototype will be foundation to develop real applications with large-scale RDF graph data.

Categories and Subject Descriptors: H.2 [DATABASE MANAGEMENT]: Physical Design, Distributed databases

General Terms: Algorithms, Design, Reliability

Keywords: Distributed, RDF, Semantic Web, Triple Store

1. INTRODUCTION

RDF is a data model for representing labeled directed graphs and is used as an important building blocks of semantic web. RDF can also be extended easily to ontologies, such as RDFS and OWL, where these ontologies provide a means to define vocabularies specified to some domain, schemas, and relations between elements of vocabulary. Recently, owing to its flexibility and applicability, RDF has been popularly used in a variety of applications, such as semantic web, bioinformatics, and social networks, wherein a huge computation over a large data set is required.

For example, in semantic web large-scale graph datasets are required for complex inferencing; in life science many projects are accumulating DNA sequence and protein interaction network datasets; and the social networks area needs massive sample data sets for complex mining of relations between people (and groups). In these days, volume of graph data generated from these applications reach tens of petabytes. However, existing systems are not sufficient to deal with these large-scale RDF graph datasets due to the following two reasons: limited I/O bandwidth and NP-Completeness of various graph problems (e.g. subgraph isomorphism).

As a practical approach to overcome these problems above, we use the server-cluster approach, where the large RDF data is partitioned and distributed over multiple servers, and the given complex subgraph query is dissected/delivered into multiple sub queries and processed over multiple servers in a distributed and parallel manner. The approach appears to be simple, but there are some challenges on the way of distribution: (i) how to distribute RDF data (ii) How to distribute the computation (and also gathering the results)

In this demonstration, we present a RDF subgraph processing system, which enables Scalable, Parallel / Distributed Evaluation of large-scale RDF data (SPIDER). It aims to store and process large-scale RDF graph datasets. In order to store and process a huge amount of RDF graph data, we use a distributed file system and employ the MapReduce framework[5] respectively.

2. SPIDER

This section gives an overview the design and implementation issues of SPIDER. In Section 2.1, we briefly presents the system architecture. In Section 2.2, we present how SPIDER performs graph query processing in a distributed manner.

2.1 An Overview of System Architecture

In order to provide extensible storage and scalable processing capability, SPIDER is based on not only distributed file system (DFS) but also distributed processing technique.

In order to focus on distributed subgraph processing, we make use of Hadoop[1] and HBase[4] (i.e., a clone of Bigtable[3]) rather than implementing new distributed framework that supports DFS and distributed and parallel processing.

The SPIDER system consists of master and triple servers. The master is an entry point for external applications and manages the triple servers. The triple servers keep assigned triples; a triple is a basic data element of RDF data. Besides, triple servers play either mappers or reducers in MapReduce.
framework while processing RDF graph queries. Each triple server stores RDF data by making use of column-oriented store[6]. In terms of semi-structured data, RDF data are very likely to be sparse because each resource has various attributes (or predicates), and it is designed for integrating heterogeneous schema dataset. Therefore, RDF data is suited for column-oriented store which is effective in handling sparse and wide tables.

In addition, the SPIDER system has two modules, which are (1) the graph loader and (2) the graph query processor. The graph loader plays a role to import the a large amount of RDF data from distributed file system to SPIDER store system. The graph query processor plays a role to search subgraphs from their own local disks and to refines candidates of final results.

2.2 RDF Graph Query Processing

Overall, with map-reduce processing the graph query processor searches subgraphs matched to a given query. Although MapReduce paradigm makes query processing of SPIDER scalable, it incurs several challenges. One of the most crucial problems is that each triple server is not aware of entire RDF data. In other words, a triple server has only partial datasets. Although SPIDER uses a distributed file system, triple servers do not communicate one another during processing because MapReduce is based on shared-nothing architecture that does not restrict communication during processing. Hence, each triple server searches subgraphs from data sets stored on its own local storage. As a result, it cannot discard even partial matched subgraphs because each triple server cannot determine if they belong to final results. For this reason, SPIDER needs further refinement processing to find compositions of subgraphs that form complete subgraphs. And then, SPIDER checks if these complete subgraphs are matched to the given query, and then it produces final results. Note that the refinement processing incurs severe computation and communication costs that deteriorate the performance of query processing. Therefore, we need to suppress the refinement processing. We found that preserving the locality of RDF graph data in regards to their connectivity reduces the the number of the refinement processing. For this reason, we sort stored triples in order of their URIs so as to give small locality to RDF data. Also, we remain this problem for future work about giving further locality of stored RDF data.

Fig. 1 shows a RDF graph query processing architec-

ture. In the figure, a SPIDER cluster composed of three nodes (i.e., \( N_1 \), \( N_2 \) and \( N_3 \)) is processing queries (\( Q_1 \), \( Q_2 \) and \( Q_3 \)). In the first, map operations retrieve the partial matched subgraphs. After this step, mappers pass retrieved subgraphs to reducers. And then, nodes \( N_1 \), \( N_2 \) and \( N_3 \) perform reduce operations for queries \( Q_1 \), \( Q_2 \) and \( Q_3 \) respectively. During this step, each of them combines partially matched subgraphs and check them if combined subgraphs matched to the given query.

3. DEMONSTRATION SCENARIO

To illustrate how the SPIDER is scalable, we will demonstrate the SPIDER running on a cluster. In this demo, we use the 100 million triples generated by the Berlin SPARQL Benchmark[2].

SPIDER provides a graphic user interface (GUI) that features the graph loader and the graph query UI. The graph loader UI enables users to import a RDF dataset from a local file system or DFS. The graph loader UI enables users to give an instance name to the loaded graph data.

The graph query UI provides a text form to enable users to submits subgraph patterns defined by SPARQL query language. When a user submit a query, the query processing starts with map-reduce operation. After query processing, the graph query UI outputs results as a list of triples.

4. EXPERIMENTS

We built a SPIDER cluster on 6 machines (consist of 2 Xeon Quad-Core, 2 Core2 duo, 2 Pentium 4). We conduct experiments for the graph loading and the graph querying with basic graph pattern in SPARQL. The graph loading takes 16 minutes to load 100 million triples; 10k triples/s. The basic graph pattern takes from several seconds to tens of seconds in terms of the given query size. If you want further information, contact http://dbaerver.korea.ac.kr/projects/spider.

5. ACKNOWLEDGMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (No. R01-2008-000-20564-0).

6. REFERENCES


