MapSQ: A Plugin-based MapReduce Framework for SPARQL Queries on GPU

Jiaming Song^{1,5}, Xiaowang Zhang^{1,5,*}, Peng Peng³, Zhiyong Feng^{2,5}, and Lei Zou⁴

¹School of Computer Science and Technology, Tianjin University, Tianjin 300350, China

²School of Computer Software, Tianjin University, Tianjin 300350, China

³College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China

⁴Institute of Computer Science and Technology, Peking University, Beijing 100871, China

⁵Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin 300350, China

*Corresponding author

ABSTRACT

In this paper, we present a plugin-based framework (MapSQ) with three parts for SPARQL queries utilizing high-performance of GPU to accelerate answering in a convenient way. Selector chooses suitable join order according to characteristics of data and queries. Executor answers subqueries and returns intermediate solutions and GPU Computing obtains the join result of intermediate solutions through MapReduce. Finally, we evaluate MapSQ built on gStore and RDF-3X on the LUBM benchmark and YAGO datasets (over 200 million triples). The experimental results show that MapSQ significantly improves the performance of SPARQL query engines with speedup up to 33.

KEYWORDS

RDF, SPARQL, Parallel Computing, MapReduce, GPU

1 INTRODUCTION

Queries over RDF data (i.e., a triple) can be expressed as SPARQL which is the standard query language over an RDF graph (a set of triples). To process SPARQL queries, there emerge many engines with various kinds of storage methods and query approaches [1]. However, those SPARQL query engines based on different optimizations bring difficulties to essential improvement of performance in processing large-scale data.

As an efficient parallel processing way, graphics processing units (GPUs) are pervasively applied and provide a higher level of speedup by executing multiple threads simultaneously. A lot of GPU-based systems have been proposed for large-scale data processing, such as GCMR [3]. However, there is few open researches on utilizing GPU to query RDF graph. In addition, the cost of transmission between CPU and GPU has to be taken into account [4].

In this paper, we present a plugin-based MapReduce framework for SPARQL query processing on GPU. This framework contains three parts, namely, *Selector*, *Executor*, and *GPU Computing*. We apply Selector to choose the best underlying engine and suitable join order according to characteristics of data and queries. Furthermore, Executor is used to answer subqueries and return intermediate

solutions by employing SPARQL engines. Finally, GPU is applied to compute the join of intermediate solutions through MapReduce on GPU. We evaluate four popular kinds of queries in MapSQ on gStore and RDF-3X over the LUBM and YAGO datasets. The experimental results show that GPU in MapSQ essentially improves the performance of SPARQL engines with speedup up to 33.

2 FRAMEWORK OF MapSQ

The framework of MapSQ is shown in Figure 1.

Figure 1: Framework of MapSQ

MapSQ consists of three main parts: *Selector*, *Executor*, and *GPU Computing*.

Selector. Selector also contains three modules: *Analyzer*, *Query Engine Selector*, and *Join Decoder*.

- The module of Analyzer provides arguments for engine selecting and join-order decision making by counting the distinct predicates of the RDF data and classifies the shape of SPARQL queries. In fact, SPARQL query engines are mostly sensitive to the shape of SPARQL queries [1]. For instance, RDF-3X [5] is more suitable for triangle queries than gStore [2] due to the VS-tree index of gStore.
- The module of Query Engine Selector, based on arguments from Analyzer, chooses a suitable engine to improve performance in an adaptive way.
- The module of Join Decoder, based on arguments from Analyzer, splits a SPARQL query into subqueries to be joined in the GPU Computing in order to find a best strategy for GPU join. In the parallel processing on GPU, load balancing and the cost of transmission between host memory and device memory are two major factors of the performance. So

This paper is published under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.

WWW '18 Companion, April 23-27, 2018, Lyon, France

© 2018 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC BY 4.0 License.

ACM ISBN 978-1-4503-5640-4/18/04.

<https://doi.org/10.1145/3184558.3186939>

the best place for a GPU join is where the subqueries have the similar small amounts of the partial results. For instance, consider Query2 of LUBM shown in Figure 2. Because gStore has powerful processing capability of the subgraphs with less selectivity, Join Decider assigns the subqueries of this type to the underlying engines as more as possible.

Figure 2: An example of join deciding for gStore

Executor. The part of Executor provides SPARQL API to answer subqueries and then obtains all intermediate solutions by employing SPARQL engines.

GPU Computing. The module of GPU Computing provides a MapReduce-based Join algorithm utilizing GPU for parallel joining. Note that the tasks can be decomposed and assigned to several streams. In this way, the proposed algorithm can improve the performance of joining because the specified stream parallel processing of GPU is suitable for executing in the pipeline via MapReduce. For instance, the copy and executions of kernel are overlapped so that the time used is reduced as shown in Figure 3.

Figure 3: An example of multi-stream overlapping

3 EXPERIMENTS AND EVALUATIONS

Hardware Setup. Our experiments are performed on a PC equipped with 64GB memory, 3.4GHz Intel CPU, a NVIDIA Tesla M40 GPU, which has 24GB device memory, 288 GB/sec bandwidth and is clocked at 1.7Hz. The operating system is Ubuntu 14.04. We use MapSQ-3X and MapSQ-g by employing RDF-3X [5] and gStore [2] within MapSQ.

Datasets. In the first experiment, LUBM400 (55.26 million triples, 6.71GB) to LUBM1600 (221.30 million triples, 27.07GB) are used to compare the performance of RDF-3X and MapSQ-3X. Because of memory constraint for loading datasets in gStore, we omit the evaluation of gStore and MapSQ-g over LUBM. In the second experiment, YAGO (200.73 million triples, 13.53GB) is used to evaluate the performance of gStore, MapSQ-g, RDF-3X, and MapSQ-3X.

Query Performance. We defines the SPARQL queries classified into four categories: linear (L), triangle (T), star (S), and snowflake (F) in our experiments. Figure 4 shows the RDF3X performs better with MapSQ (speedup of T1 over LUBM1600 up to 32.42) and MapSQ makes it more scalable for larger LUBM datasets. Similarly, Figure

5 shows the improvement provided by MapSQ over YAGO datasets. Especially the speedup of L1 is up to 33.01.

Figure 4: Query runtimes for LUBM datasets

Figure 5: Query runtimes for YAGO dataset

4 CONCLUSIONS

In this paper, we proposed a plugin-based framework to take advantage of high-performance of GPUs for supporting large-scale RDF data querying in an efficient and simple way. We think that this proposal would provide an idea to process large-scale RDF data by employing new processing units such as GPU and FPGA.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (2016YFB1000603) and the National Natural Science Foundation of China (61672377, 61502336).

REFERENCES

- [1] Abdelaziz I., Harbi R., Khayyat Z., and Kalnis P. 2017. A survey and experimental comparison of distributed SPARQL engines for very large RDF data. *PVLDB* 10(13): 2049–2060.
- [2] Chen L., Zsu M.T., Zou L., Mo J., and Zhao D. 2011. gStore: Answering SPARQL queries via subgraph matching. *PVLDB* 4(8):482–493.
- [3] Guo Y., Liu W., Voss G., and Mueller-Wittig W. 2014. GCMR: A GPU cluster-based MapReduce framework for large-scale data processing. In *Proc. of HPCC & EUC* '14. 580–586.
- [4] Kaldewey T., Lohman G., and Mueller R. 2012. GPU Join processing revisited. In *Proc. of EDBT'12*. pp.55–62.
- [5] Weikum G. and Neumann T. 2008. RDF-3X: A RISC-style engine for RDF. *PVLDB* 1(1):647–659.