Performance Evaluation of Large Table Association Problem Implemented in Apache Spark on Cluster with Angara Interconnect

Alexander Agarkov and Alexander Semenov

JSC NICEVT, Moscow, Russia
{a.agarkov,semenov}@nicevt.ru

Abstract. In this paper we consider an association problem with constraints for two dynamically enlarging tables. We consider a base full association algorithm and propose a partial association algorithm that improves efficiency of the base algorithm. We implement and evaluate the algorithms in Apache Spark for a particular case on the cluster with Angara interconnect.

Keywords: association problem · dynamically enlarging tables · Apache Spark · Angara interconnect · performance evaluation

1 Introduction

In the recent years data intensive applications have become widespread and appeared in many science and engineering areas (biology, bioinformatics, medicine, cosmology, finance, social network analysis, cryptology etc.). They are characterized by a large amount of data, irregular workloads, unbalanced computations and low sustained performance of computing systems. Development of new algorithmic approaches and programming technologies are urgently needed to boost efficiency of HPC systems for similar applications, thus enabling advancing of HPC and Big Data convergence [10].

In the paper we consider an association problem with constraints for two dynamically enlarging tables. We have two large tables and an ordered set of rule groups which determine associations between entries from the first table and the second table. When two table entries compose an association by a rule in the current rule group, then these entries must be excluded from association process for the following rule groups. Each entry is associated with other entries from the both tables directly or indirectly through the other associations. It is needed to determine the association type and to list of the associated entries for each entry. Tables are dynamically enlarging, the goal is to improve potential performance of the association process by using of the associations, built on the original tables.

Apache Spark [1] is a popular open-source implementation of Spark. Spark [12] is a framework which optimizes programming and execution models of MapReduce [7]. Current implementation of Apache Spark can not efficiently use advanced features (e.g. RDMA) on clusters with high-performance interconnects. Researchers from Ohio State Univercity proposed a high-performance RDMA-based design for accelerating the Spark framework [9, 8]. Chaimov et al. from Cray ported and tuned Spark on Cray XC systems developed in production at a large supercomputing center [6]. The Mellanox company presented an open-source Spark RDMA implementation [2]. We consider a high-speed Angara interconnect [4] as a target of Spark optimization. But in the current work we run Apache Spark through the TCP/IP interface on the Angara interconnect.

In the paper we describe a base full association approach to the problem, propose a partial association approach that improves efficiency of the base approach, implement corresponding algorithms using Apache Spark and present evaluation results on a cluster with the Angara interconnect.

2 Table Association Problem

We consider two large *tables* with M rows. The tables have identical structure, each table has N fields. The table unique key consists of all N fields of the table.

We consider an ordered set of rule groups, which determine associations between entries from the first table and entries from the second table:

- *Rule* is a set of fields, which are used to compare two table entries. It is required to build associations between the tables: to find matches between different table entries by the rule.
- *Group* is a set of rules; rules of a group are applied to the table entries independently of each other. When two table entries compose the association by a rule in the current rule group, then these entries are marked by the current group number and must be excluded from association process for the following rule groups.

Each table entry can be associated with one or many entries of another table. Moreover, association is a transitive relation. Associations for each entry can be classified into one of four association types: one from the first table to one from the second table, one to many, many to one and many to many (1: 1, 1: M, M: 1, M: N). Therefore each entry is associated with other entries from the both tables directly or indirectly through the other associations.

The goal of the **table association problem** is to determine the association type and to list of the associated entries for each entry.

After we build associations between the tables, K new entries are added to each table. Added entries differ from original entries by a given subset of fields. Association process is needed to repeat to make the augmented tables associated too.

Full association approach can generate associations between given tables by the mentioned set of rules from scratch. It is required to build associations between the augmented tables.

The goal of the **dynamically enlarging table association problem** is to improve potential performance of the full association approach on the augmented tables by using of the associations, built on the original tables.

For the sake of simplicity in the paper we consider a particular case of the problem.

2.1 Data Structure and Association Rules

In our work each table entry has 5 fields, where the first and the second fields are integer identifiers, three other fields are data fields. The unique key for every entry is all of the five fields. Each entry has a unique synthetic identifier.

In the work the considered ordered set of rule groups consists of 5 groups and 15 rules, see Table 1. Symbol '+' denotes equality requirement of the corresponding fields of the two table entries. Symbol '–' denotes that fields of the two table entries are not matched. For example, a key that determines an association between two table entries is specified for each rule and consists of the fields, that are marked with the '+' symbol.

group	rule	ID 1	ID 2	#1	#2	#3
1	1	+	+	+	+	+
2	2	+	+	+	+	−
2	3	+	+	+	−	+
2	4	+	+	−	+	+
3	5	+	+	−	+	−
3	6	+	+	+	−	−
3	7	+	+	−	−	+
4	8	+	−	+	+	+
4	9	+	−	+	+	−
4	10	+	−	+	−	+
4	11	+	−	−	+	+
5	12	−	+	+	+	+
5	13	−	+	+	+	−
5	14	−	+	+	−	+
5	15	−	+	−	+	+

Table 1. The description of the considered rules groups

The full association approach is matching each entry from the first table with each entry of the second table by each rule from the current rule group. If matching is successful then we create and store an association between the entries; the association is marked by the current group number. Entries that do not have any associations are marked by the group number six.

3 Algorithms

3.1 Full Association Algorithm

The full association algorithm consists of two stages: associations matching and transitive closure. The first stage actually implements the full association approach. All possible pairs are found by the first group of rules, then every entry that is included in the pairs is excluded from the tables. This procedure is repeated for each group of rules. The result of the stage is a set of associations (it will be graph edges) between entries (it will be graph vertices).

At the second stage the transitive closure (TC) algorithm is executed for each selected group. At first, we construct a bipartite graph. Vertices in the left vertex set are unique identifiers of the entries from the first table, in the right vertex set there are unique identifiers of the entries from the second table. There exists an edge between two vertices of the different graph parts if the association between corresponding entries has been found during the first stage of the algorithm.

Transitive closure [3] TC is performed by the following formula:

$$\begin{aligned} TC = \cup_{i=1,2..} R_i,\ where\ R_{i+1} = R_i\ join\ E, \\ R_1 = E, E - set\ of\ graph\ edges. \end{aligned} \tag{1}$$

The transitive closure is built by repetitive merging result of a join operation between previous resulting set of edges and original set of graph edges until the result is not changed, i.e. fixed point is reached. Thus, for each vertex in TC there exist vertex pairs that connect current vertex with other vertices in the connected component.

Finally, the association type of each vertex is defined (1:1, 1:M, M:1, M:N).

3.2 Partial Association Algorithm

There are original (old) tables, the associations that have been built for old tables, and there are new tables that are smaller than original. The added entries differ from original entries by the #3 field.

When new entries are added to the original tables, one can apply full association algorithm to the augmented tables from scratch. We propose a partial association algorithm to improve performance by using of associations that are built for the original tables.

The partial association algorithm is executed also in two stages: association matching and transitive closure.

It is important that added (new) entries differ from the original entries by the #3 field. The main idea of the first stage is matching only new entries of the tables for each rule with matching requirement by the #3 field, in that case there will be no associations between new and old entries. For each rule without matching requirement by the last field new and old entries must be matched, see Figure 1.

The association matching stage differs from the same stage in the full association algorithm. Each entry included in new associations must be excluded from old associations. As seen in Figure 2, if a new entry is associated with an old entry, and the association group number *new_gn* is smaller than the group number *old_gn* of the association between the old entry with another entry, then these old associations must be removed; if *new_gn* is equal to *old_gn* then the new entry should be added to the component.

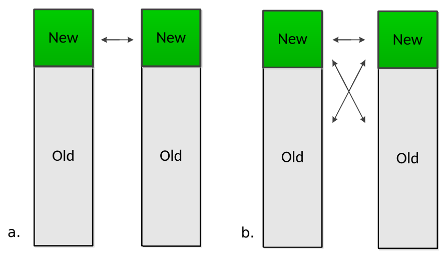

Fig. 1. Partial association algorithm. Processing a. with (b. without) #3 field matching requirement

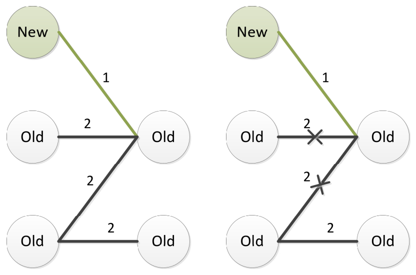

Fig. 2. Association of new entry with old entry and removing of old associations

The transitive closure stage is executed only for new associations. The resulting graph of transitive closure is combined with the old graph with invalid associations excluded during the matching stage.

4 Implementation Details

Apache Spark [1] is a popular open-source implementation of Spark. It provides programmers with an application programming interface centered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way [12]. It was developed in response to limitations in the MapReduce cluster computing paradigm [7], which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDD function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory [11]. The latest Spark program interface DataFrame [5] seems to be more efficient than the RDD interface, but in the current work we use RDD, and we suppose to use DataFrames in the next research works.

We use Java 8 and Apache Spark 1.6.1 for implementation of the full and partial association algorithms. We use RDD of `Tuple5<Long, Long, Long, Long, Long>` type for table structure representation, the sequence of types in `Tuple5` corresponds to the table fields ID1, ID2, #1, #2, #3. We attach unique identifier (`Long`) to the `Tuple5` of each entry.

After the association stage we have RDD of `Tuple2<Long, Long>` that represents association between the unique identifiers of two table entries.

4.1 Synthetic Table Generator

Synthetic table generator creates distributed random tables and works as follows. First, two identical tables of the required size are generated. Each field value is a uniformly distributed random integer number in the following intervals:

– ID1, ID2 – $[0; 10000)$,
– #1 – $[0; 1000000)$,
– #3 – $[0; 1000)$.

Value of the #2 field is a position number of the entry.

We randomly modify second table entries in order to create possibility of association between entries from the first and the second tables for each rule. We modify entry fields that are marked with the '–' symbol in Table 1. Distribution of modifications in the rules is shown in Table 2. 72% of the second table entries remain unchanged. In 2% of the table entries there are random modifications in the #3 field values. In 6% of the table entries there are random modifications in the #2 field values and so on. As a result 72% of the table entries correspond to the first rule, 2% – to the second rule, 6% – to the third rule and so on.

rule	**% in tables**
1	72
2	2
3	6
4	12
5	0.1
6	0.5
7	0.4
8	0.5
9	0.01
10	0.04
11	0.35
12	0.83
13	0.02
14	0.12
15	0.26
no associations	4.86

Table 2. Distribution of the second table modifications in the rules implemented in the synthetic table generator

We generate the augmented tables as follows. First, we add new entries to the first table, field values of each entry is a uniformly distributed random integer number in the following intervals:

– ID1, ID2 – $[0; 10000)$,
– #1 – $[0; 1000000)$,
– #3 – $[1000; 2000)$.

Value of the #2 field is a position number of the new entry in the whole table. As can be seen, the old table and the new table have different values in the #3 field.

Second, we copy augmented part of the first table to the second table and randomly modify it as well as is described in Table 2.

5 Performance Evaluation

Cluster	Angara-K1
Chassis	SuperServer 5017GR-TF
Processor	E5-2660 (8 cores, 2.2 GHz)
Memory	DDR3 64 GB
Number of nodes	36
Interconnect	Angara 4D-torus $3 \times 3 \times 2 \times 2$ 1 Gbit/s Ethernet
Operating system	SLES 11 SP4
Spark	Apache Spark 1.6.1

Table 3. System configuration of the Angara-K1 cluster

All presented results are obtained on the Angara-K1 cluster. We use 12 out 36 nodes. All Angara-K1 nodes are linked by the Angara interconnect. Russian high-speed Angara interconnect is developed in NICEVT, performance evaluation of the Angara-K1 cluster with the Angara interconnect on scientific workloads is presented in [4]. In the current work we run Apache Spark through the TCP/IP interface on the Angara interconnect. Table 3 provides an architecture and a software overview of the Angara-K1 partition.

Figures 3, 4, 5 and 6 show the comparison results of the full and partial association algorithms. The reported running times do not include reading data and writing the result to the filesystem. Table 4 presents the total table sizes in GB during implementation executing for different entry numbers.

table size, mln entries	**in memory, GB**
300	200
100	70
50	35
25	20
10	12

Table 4. Table size in GB depending on the number of table entries

The algorithm running times are shown in Figure 3, we use 8 cores per node and 8 nodes of the cluster, table size is varied. Old table size is 300 million entries, new table size is 75 million entries. The figure shows that performance difference between the algorithms grows with table size.

Strong scaling is shown in Figure 4. Old table size is 100 million entries, new table size is 25 million entries. The speedup of the full and partial association algorithms is approximately 3 on 8 nodes. Among the possible reasons of moderate performance there is a single one: Spark configuration is not optimal. Further

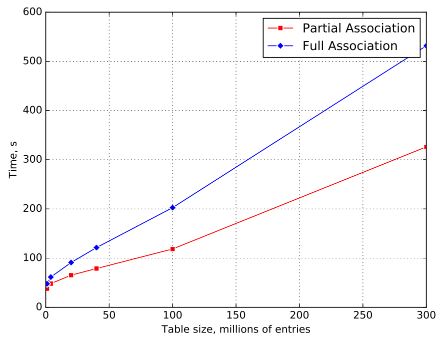

Fig. 3. Running times of the full and partial association algorithms on the different table sizes. 8 cores per each of the 8 Angara-K1 cluster nodes

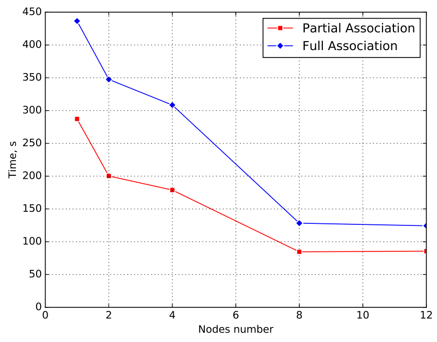

Fig. 4. Strong scaling for the old tables with 100 million entries and the new tables with 25 million entries

tuning can address the problem. Horizontal line from 8 to 12 nodes indicates that the table size is small for further performance increasing.

In Figure 5 profiling results are shown. Shaded color denotes the association matching stage (stage #1), normal color denotes the transitive closure stage (stage #2). Old table size is 300 million entries, new table size is 50 million entries. It can be seen that the partial algorithm optimizes primarily the transitive closure stage.

Running time on 4 nodes is more than two times faster than on 2 nodes, because the problem size is too large for 2 nodes and Garbage Collector occupies a significant part of the time.

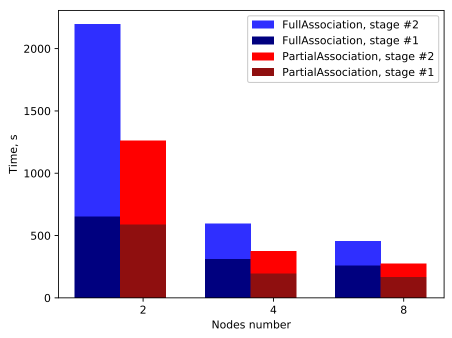

Fig. 5. Profiling results for the old tables with 300 million entries and the new tables with 50 million entries

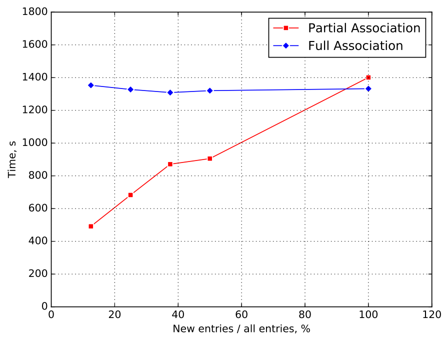

Fig. 6. Algorithm performance comparison for a various ratio of new and all table entries

The dependence of algorithm running times on the amount of new data is shown in Figure 6. We use 6 nodes, 300 million entries in the each table, the fraction of the new table entries varies from 12.5 to 100 percents of the total table size. The smaller the percentage of new data is, the faster the partial association algorithm is executed. The running time of the full association algorithm does not change, because the total amount of data does not change.

6 Conclusion

In the paper we propose the partial association algorithm for the table association problem of two dynamically enlarging tables with specific constraints. For the sake of simplicity we consider a particular case of the problem. We implement the base full association algorithm and the proposed algorithm using Apache Spark and present performance evaluation of the algorithms on the cluster equipped with the Angara interconnect. Performance of the proposed algorithm exceeds performance of the full association algorithm for a variety of data sets.

In future work we plan to make detail profiling of the implemented algorithms in terms of Apache Spark internal operations and to optimize Apache Spark on the Angara interconnect.

Acknowledgments. The work was supported by the grant No. 17-07-01592A of the Russian Foundation for Basic Research (RFBR).

References

1. Apach Spark Homepage, http://spark.apache.org/
2. Mellanox Spark RDMA, https://github.com/Mellanox/SparkRDMA
3. Abiteboul, S., Hull, R., Vianu, V. (eds.): Foundations of Databases: The Logical Level. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1995)
4. Agarkov, A., Ismagilov, T., Makagon, D., Semenov, A., Simonov, A.: Performance evaluation of the Angara interconnect. In: Proceedings of the International Conference Russian Supercomputing Days. pp. 626–639 (2016), http://www.dislab.org/docs/rsd2016-angara-bench.pdf
5. Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A., et al.: Spark SQL: Relational data processing in Spark. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 1383–1394. ACM (2015)
6. Chaimov, N., Malony, A., Canon, S., Iancu, C., Ibrahim, K.Z., Srinivasan, J.: Scaling Spark on HPC systems pp. 97–110 (2016)
7. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: Proceedings of the 6th Conference on Symposium on Opearting Systems Design and Implementation - Volume 6. OSDI'04, USENIX Association, Berkeley, CA, USA (2004)

8. Lu, X., D., S., Gugnani, S., Panda, D.: High-performance design of apache spark with rdma and its benefits on various workloads (December 2016)
9. Lu, X., Rahman, M.W.U., Islam, N., Shankar, D., Panda, D.K.: Accelerating spark with rdma for big data processing: Early experiences. In: Proceedings of the 2014 IEEE 22Nd Annual Symposium on High-Performance Interconnects. pp. 9–16. HOTI '14, IEEE Computer Society, Washington, DC, USA (2014)
10. Reed, D., Dongarra, J.: Exascale computing and Big Data: The next frontier. Communications of the ACM 57(7), 56–68 (2014)
11. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. NSDI'12, USENIX Association, Berkeley, CA, USA (2012)
12. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. HotCloud 10, 7 (2010)