An Adaptive Approximate Algorithm For Join

Oxana Dolmatova

Saint Petersburg State University

oxana.dolmatova@gmail.com

Advisor: Boris Novikov

Abstract

This paper describes the initial state of development adaptive approximate algorithm for join combining three modifications of the traditional techniques

The main result of the research will be the new algorithm suitable for pipelined inputs, a cost model for this algorithm as well as experimental evaluation to estimate the accuracy of cost model and identification the various properties of the algorithm.

1 Introduction

In this paper we consider query evaluation in the distributed heterogeneous systems, real-time systems, mobile systems or environment with data [1]of varying intensity. In all these cases we would like to perform queries with in predictable response time and perhaps not exactly because of lack of resources or time constraints.

In contrast with traditional databases this context creates additional challenges.

As a consequence it is clear that the approximate algorithms for computationally intensive operations are unavoidable in modern environments.

In the context of complex queries, join is one of the most important and intensively used operations in the users' search operations.

There are three types of algorithms in traditional databases for join and all three algorithms are designed to obtain the exact answer. Adaptive execution of join will combine all the benefits of the existing standard algorithms, whether sorted inputs, the presence of permanent indexes for one or two threads or a different input sizes of the unsorted pipelines.

It is significant that we call the input data - pipelined data, there is one difference between stream and pipelined data. To use our adaptive algorithm we have to know the right and left streams' sizes to choose the right strategy. Nevertheless the processing of incoming objects will be the same as work with streaming data, which gives an advantage in the sense that we can pipeline a result, even without getting all the data.

Also a new algorithm will be approximate; it means that the quality of the final result depends on the amount of allocated resources. Or rather, in this case,

the term approximate means incomplete result that is some of relevant output tuples may be missed in the result and all pairs that will be included definitely will satisfy the join predicate.

The exact definition and method of measuring the quality of result we are planning for the future work. Considering quality very tentatively, it looks like the precision. We use the term precision in the usual sense: it is the ratio between obtained tuples and all the pairs that fit the predicate.

The abstract concept of term "resource" may include several different performance measures of a query execution. Usually the most important is the execution time (either CPU or elapsed), amount of I/O, or, in a distributed mobile environment, the battery energy.

That is, with limited resources, the answer will be incomplete. However, those properties provide control trade-off between quality and cost.

The following section overviews the different variation of join algorithms such as adaptive and similarity schemes. Modifications of the three standard algorithms for join you can find in part 3. The section 4 introduces a small description and main idea of the new join technique. The last part of this paper offers our conclusions and certain ideas for future work.

2 ReleatedWork

The inspiration of this work was the work of [5]. It formed the basis of the main idea of the new algorithm for the approximate join.

Author is building an effective technique for adaptive query, based on the standard algorithms, and taking the best of all of them. In contrast to this paper the algorithm is not designed to work with the streams and quality management through resources.

An adaptive algorithm for similarity join evaluation can be found in [6].

The proposed technique is designed similar to those and suitable to the pipelined architecture. It means that the algorithm is incremental and begins returning results even before streams have completely been consumed.

Proceedings of the Ninth Spring Researcher's Colloquium On Database and Information Systems SYRCoDIS, Kazan, Russia, 2013

This method is an aggregation of the best from SHJoin [9] and SSJoin [2].

An adaptive nested loop-based algorithm for stream input was presented in [1]. The main advantage over traditional join techniques is that this algorithm can start producing join results as soon as the first input object are available, thus improving pipelining by smoothing join result production and by masking source or network delays.

Then the authors present a sophisticated version of the algorithm suitable for multiple input streams. Thereby this research exploits temporary delays when new data is not available for processing data obtained.

A stream input for join algorithm with emphasis on the limited internal memory was discussed in [3].

The authors consider the task of sliding windows join. In case of resource shortage, tuples have to be dropped before they expire thus this algorithm can be called approximate. The same authors propose a way to estimate the error of the executed operation.

The join algorithm suitable for integration was introduced in [7]. The authors propose the trade-offs between the completeness of a join result, and its execution efficiency: users can choose a faster execution, at the price of missing more matches, resulting in a lower result completeness; or a more complete join result, at the cost of lower performance.

That proposal looks like one of our future goals. At the same time, the term adaptive means no variation between the three standard algorithms for join, and aggregation exact join and similarity join, what distinguishes this paper from our own.

The [10] introduces the adaptive hash join algorithm in connection with a problem of multiuser environment. Authors present a modified join technique that is designed to work with dynamic changes in the amount of available memory.

A general aim of this work is to regulate resource usage and to provide the way that allows query execution to run concurrently with other applications.

Therefore there is a wide range of very different algorithms for the join operation. However almost all of them are tailored for either similarity join or to join for stream input. As well as some of algorithms represent an adaptive technique for the exact query execution.

While the ultimate goal of this work is the detailed approximate adaptive algorithm for join.

3 Traditional Basic Algoritms

Detailed description of the following standard algorithms with history, options and optimization and cost models can be found in [5].

Here we will describe the properties of modified algorithms that will be the basis of a new adaptive algorithm.

3.1 Hash-based join

We choose the symmetric hash join modification for this section. As introduced in [9] this method has a similar behavior to the standard hash join with some peculiarities.

First of all the normal join uses only one hash table to calculate its results, while the symmetric hash join uses two tables, one for each input. Second, the normal join is a blocking operator and as a consequence not suitable for pipelined architectures.

To begin yielding results it has to entirely build a hash table over one of the inputs, meaning that no result is returned before at least one of the tables has been completely read. The symmetric hash join on the other hand is a non blocking operator. The two hash tables are built in parallel while reading the objects from both inputs.

The algorithm is able to return result as soon as first objects are coming from both streams. It does not need to wait for complete input.

3.2 Nonblocking merge-based join

That is probably the easiest algorithm to modify to approximate mode. There are sorted input pipelines, and their size. For the approximate performance we stop the execution when the resources allocated to the operation (for example, time) running out.

It should be noted that if the input data are not sorted, this algorithm cannot be used because it is not possible to sort the pipelined data.

3.3 Nonblocked nested loop-based join

Since we have a normal, two-input, join, respectively, there are also two cycles in nested loop. Here it is necessary to consider the following cases.

1. There are the pipelines without any index and the one of the threads is much smaller than the other and we have the sufficient amount of resources. Then, for each obtained object in the big pipeline algorithm takes a loop with already read data from small input, until allocated time runs out.
2. There are two small pipelines and we have sufficient amount of resources to read and process it. Hence, it will be exact query execution with exact standard nested loop
3. There are two pipelines and its sizes are not allowed to read pipeline completely because of lack of resources. Then the algorithm turn scans each obtained object by comparing it with those of the objects from another pipeline, which we have already obtained. When the resources run out, the algorithm will stop, and we will get an approximate result.

4 Description of the new algorithm

The choice of the strategy depends on the properties of the input data: how much different in size input pipelines and how many pipelines are sorted, and how many resources were allocated for the join execution.

If both pipelines are sorted by the join attribute, then the new algorithm is similar to the non blocked merge join, referred to in 3.2.

If only one of the pipelined inputs is sorted, the algorithm behaves like an aggregation from the merge join and the nested loop. In this case an algorithm has different behaviors depending on the input size and resources available for the operation:

- The sorted input fits into the memory which allocated for join execution. And we have sufficient (that term will be specify in cost model in the future work) amount of processing resources which allows read all this small input. Then algorithm reads this input and after that reads as many objects from other inputs as can be located in the remaining memory. The algorithm sorts read objects from second input and makes sort-merge join of those two sets of objects. After that if we still have the resources the algorithm reads other block of objects from second and sorts it etc.
- The sorted inputs can fit into the memory but we haven't sufficient amount of processing resources. The algorithm receives part of first input and part of second input, sorts it and performs sort-merge join of those two sets.
- Any of inputs cannot fit into the memory and we have the sufficient amount of resources. Then the algorithm reads several blocks from sorted input and block from unsorted input sorts this block and makes sort-merge join. As long as there are resources the technique obtains the blocks of both inputs and makes merge sort.

If any of pipelines are not sorted the algorithm's behavior looks like in the previous case. The difference is in the number of objects and loops which algorithms makes and reads in given amount of resources.

5 Conclusion and future work

We are going to implement our algorithm with assumption that the algorithm of join will be used in the context of [8].

The optimization is essential for any high-performance querying system. Actually, the task of a query optimizer is to choose an algebraic expression of minimal execution cost among several equivalent expressions. Because of any high-quality optimizer is inevitably a cost-based one and, hence, the cost model is one of the critical core components of the optimizer we are going to develop a cost model for our new algorithm.

As the query evaluation is approximate, the quality of the output can be lower than the exact. Hence it is necessary to estimate an adequate quality of the result which will determine the accuracy of the obtained answer.

As a result it will be possible to control trade-off between resource and quality.

And at the end we are going to conduct the experiments for both estimate the accuracy of the constructed cost model and to compare the performance of different implementations approximate join with our algorithm.

For example, it will be interesting to see how the number of found pairs of answer is depending on the initial data: sorted and unsorted input pipelines, small and large amount of input data, the large difference in the amount of input data or the comparable size of the right and left thread.

Such experimental results can show that sensible savings in join execution time can be achieved in practice; at the expense of a modest reduction in result completeness. Those experiments will be assigning to determine the characteristics and properties of the new algorithm.

References

[1] Mihaela A. Bornea, Vasilis Vassalos, Yannis Kotidis, Antonios Deligiannakis: Adaptive Join Operators for Result Rate Optimization on Streaming Inputs.IEEE Trans. Knowl. Data Eng. (TKDE) 22(8):1110-1125 (2010)

[2] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarityjoins in data cleaning. In L. Liu, A. Reuter, K.-Y. Whang, and J. Zhang,editors, ICDE, page 5. IEEE Computer Society, 2006.

[3] Abhinandan Das, Johannes Gehrke, Mirek Riedewald: Approximate Join Processing Over Data Streams. SIGMOD 2003:40-51

[4] Oxana Dolmatova, Anna Yarygina, Boris Novikov: Cost Models for Approximate Query Evaluation Algorithms. DB&Local Proceedings 2012:20-28

[5] Goetz Graefe: New algorithms for join and grouping operations. Computer Science - R&D 27(1): 3-27 (2012)

[6] Roald Lengu, Paolo Missier, Alvaro A. A. Fernandes, Giovanna Guerrini, Marco Mesiti: Symmetric set hash join: A pipelined approximate join algorithm, http://unina.stidue.net/Universita'%20di%20Genova/GuerriniG/reports/sshjoinTR.pdf

[7] Paolo Missier, Alvaro A. A. Fernandes, Roald Lengu, Giovanna Guerrini, Marco Mesiti: Data Quality support to on-the-fly data integration using Adaptive Query Processing. SEBD 2009: 213-220

[8] Boris Novikov, Natalia Vassilieva, Anna Yarygina: Querying big data. CompSysTech 2012: 1-10

[9] A. N. Wilschut and P. M. G. Apers. Dataflow query execution in a parallelmain-memory environment. In PDIS, pages 68{77. IEEE Computer Society,1991

[10] Hansjörg Zeller, Jim Gray: An Adaptive Hash Join Algorithm for Multiuser Environments. VLDB 1990: 186-197