Information Systems 58 (2016) 105–125

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

PaMeCo join: A parallel main memory compact hash join

Steven Begley*, Zhen He, Yi-Ping Phoebe Chen

Department of Computer Science and Information Technology, La Trobe University, VIC 3086, Australia

ARTICLE INFO

Article history:
Received 31 August 2015
Accepted 5 October 2015
Recommended by: Philippe Bonnet
Available online 6 November 2015

Keywords:
In-memory databases
Hash join
Memory constrained
OLAP
Column store

ABSTRACT

This paper presents a memory-constrained hash join algorithm (PaMeCo Join) designed to operate with main-memory column-store database systems. Whilst RAM has become more affordable and the popularity of main-memory database systems continues to grow, we recognize that RAM is a finite resource and that database systems rarely have an excess of memory available to them. Therefore, we design PaMeCo to operate within an arbitrary memory limitation by processing the input relations by parts, and by using a compact hash table that represents the contained tuples in a compact format. Coupled with a radix-clustering system that lowers memory latencies, we find that PaMeCo can offer competitive performance levels to other contemporary hash join algorithms in an unconstrained environment, while being up to three times faster than a high-performing hash join when memory constraints are applied.

© 2015 Elsevier Ltd. All rights reserved.

1. Introduction

Column-store databases remain a hot topic for database researchers, with existing systems (e.g. [1–4]) showing desirable performance characteristics in scenarios such as online analytical processing (OLAP), which encompasses fields such as business analytics, marketing, budgeting, financial reporting, and other business intelligence systems. Column-store systems are also making inroads into other domains previously considered as weak points for column-store systems, such as online transaction processing (OLTP) [5]. We find most column-store systems being tailored for running as main memory database systems (due to the present-day affordability of RAM), avoiding the latencies associated with secondary storage systems. This has led to researchers to look at new ways of optimizing database algorithms to be better suited to running in a main memory scenario.

The relational join, being one of the most important database operators, can also be one of the most processor and memory resource intensive operations. A popular approach to performing hash join operations on main memory column-store databases is through the use of the radix-cluster algorithm [1], due to its ability to overcome memory access stalls by using cache-friendly data structures. However, many existing join algorithms make an assumption that there is an unlimited reserve of temporary memory available, which can prove problematic in situations where there is insufficient free temporary memory to fully build the input relations into hash tables (or other data structures) during the join operation. For example, the radix join (a hash join utilizing the radix-cluster algorithm) partitions and builds the input relations before executing the join phase, therefore needing temporary memory at least the size of the input relations themselves. To compound this situation, a database system may perform multiple queries simultaneously, and thereby requiring their aggregate memory space. When main memory is exhausted the virtual memory system pages data to disk, significantly lowering the performance of the join operation.

* Corresponding author.
E-mail addresses: skbegley@students.latrobe.edu.au (S. Begley), z.he@latrobe.edu.au (Z. He), phoebe.chen@latrobe.edu.au (Y.-P. Chen).

http://dx.doi.org/10.1016/j.is.2015.10.004
0306-4379/© 2015 Elsevier Ltd. All rights reserved.

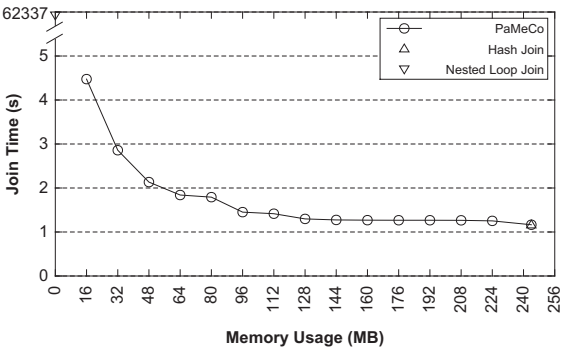

Fig. 1. Join algorithm temporary memory usage and performance comparison.

An existing join algorithm that requires virtually no temporary memory resources is the nested loop join, which compares every tuple in the outer relation with every tuple in the inner relation. However, unlike the hash join, the nested loop join does not attempt to prune the number of comparisons, and therefore typically gives poor performance when workloads are not small. Ideally, we desire a join algorithm that can offer the performance benefits of a hash join, but with the ability to constrain the temporary memory overheads as needed, even to a level approaching that of the nested loops join.

Fig. 1 shows the performance achieved by the Parallel Memory Constrained Join (PaMeCo Join) proposed by this paper. It demonstrates that PaMeCo can achieve performance similar to the ideal mentioned above, namely being able to largely retain the performance of the hash join with very tight memory constraints. Even at a low memory limit of 16 MB (approx. 7% of the temporary memory needed by the hash join in this example), PaMeCo outperforms the nested loop join by six orders of magnitude. In this demonstration both relations had 16 M tuples, and each tuple is a key/value pair totaling 8 bytes in size.

A simple and intuitive way to impose a memory limit on the hash join is to process the input relations in blocks (i.e. consecutive groups of tuples), in a manner similar to that of a Block Nested Loops join [6] with a hash join performed between the blocks. However, this approach leads to an increased total number of hash table building and comparison operations because one pass through the entire inner relation is required for each block of the outer relation.

Our aim is to process larger blocks of the outer relation while still obeying a memory limitation and retaining the performance benefits of a hash join. We have achieved this by designing and implementing the Parallel Memory Constrained (PaMeCo) join algorithm, a hash join that utilizes a compact hash table that works within an arbitrary memory constraint. The results show that in a memory constrained environment, PaMeCo can outperform a naively memory-constrained version of a hash join algorithm.

PaMeCo achieves this result as follows. Firstly, during the *build* phase, PaMeCo uses compression to reduce the size of the hash table containing the tuples of the outer relation, and therefore allowing larger blocks of the outer relation to be stored in temporary memory. This leads to fewer passes through the inner relation, thereby reducing the overall join cost. The compression algorithm exploits the nature of the hash table process itself to get the compression and decompression for very little overhead.

Secondly, PaMeCo uses a radix-clustering system that can operate within a limited memory space, which helps to avoid some of the cache miss latencies that can occur during the build and probe phases of the join process.

Thirdly, like some other hash join algorithms [7,8], PaMeCo employs a histogram to manage the building of the hash table. PaMeCo improves upon the efficiency of this histogram by allocating the memory just once during the join process, and reusing the space allocated for the histogram for multiple purposes.

Finally, PaMeCo takes advantage of the multi-cored topology that is typical of current commodity processors by utilizing thread-level concurrency. It achieves this functionality whilst remaining mindful of operating within a given memory constraint. In particular scenarios PaMeCo can avoid some of the memory and performance overheads that may be incurred though multithreading by providing thread-level concurrency that is free of locking mechanisms.

The objective of this paper is not to design the fastest join algorithm in unconstrained memory environments. Instead, our interest is in how little performance degradation we can incur as the system memory becomes more constrained. We hope that the core ideas presented in this paper may be of use to database researchers and complement the development of other join algorithms.

In summary this paper makes the following key contributions:

1. Identifies the importance of making join algorithms for main memory databases memory constrained. This is in contrast to most existing literature for main memory joins, which assume unlimited available temporary memory.
2. Proposes a memory constrained join algorithm called PaMeCo. The algorithm processes the source relations in a manner similar to a block nested loops join, but utilizes a compact hash table to maximize the size of the outer blocks.
3. Proposes a method of parallelizing the shared resources of the PaMeCo algorithm whilst remaining mindful of memory constraints and performance burdens. A specialized case of PaMeCo achieves thread-level concurrency without the use of conventional locking mechanisms.
4. Finally, a detailed empirical study of the performance of PaMeCo vs. a naively memory-constrained version of a competitive hash join algorithm (based on the work of [7]) was conducted. The results show the superiority of PaMeCo in a variety of situations.

This paper extends the work of our earlier paper [9]. There, we presented the MCJoin algorithm and compared its performance against a hash join algorithm using single-threaded workloads in a memory-constrained environment. In this paper, we examine the challenges of

Download English Version:

https://daneshyari.com/en/article/397233

Download Persian Version:

https://daneshyari.com/article/397233

Daneshyari.com