Dynamic Load Balanced Join Processing

Kian-Lee Tan Hongjun Lu

Department of Information Systems and Computer Science
National University of Singapore
Lower Kent Ridge Road
Singapore 0511

Abstract

In this paper, we revisit the problem of processing joins in shared-nothing systems. In such systems, a join query is usually split into a set of tasks that are allocated to the nodes in the system where the tasks could be executed concurrently and independently. While parallel processing could greatly reduce the completion time of a join operation, the system performance may degrade because of load imbalance across the nodes caused by data skewness in the relations. Most of the previous studies addressed this issue by *static* load balancing. We proposed here two *dynamic* load balancing join strategies and compared their performance with two static algorithms. The result of our study shows that dynamically balancing the join load is not only feasible but also provides good system performance.

Keywords: Shared-nothing systems, join, load balance, dynamic, data skew.

1 Introduction

A *shared-nothing* system comprises a set of processing nodes interconnected through a communication network. Each node has its own local memory and disk drives. A processor at one node has no direct access to memory or disks of other nodes. Data sharing among processors is realized by some message passing mechanisms. To facilitate parallel processing, data, or relations are typically, partitioned horizontally across the nodes. As such architecture can be scaled up to hundreds and probably thousands of processing nodes, it becomes one of the major parallel database system architectures [1]. Compared with other parallel database architectures, such as shared-memory system, one major disadvantage of shared-nothing architecture is that, load balancing among the processing nodes becomes more important but more difficult.

Consider the parallel processing of the expensive relational *join* operation that has received much attention in recent years [3, 5, 6, 10, 9, 12, 13, 15, 16]. The join is first split into a set of independent tasks[1] which are then allocated to the nodes to be processed concurrently and independently. However, because of data skew, the allocation may result in unbalanced load across the nodes in the system, hence degradation of performance [8, 14].

While many algorithms have been proposed to handle data skew, these methods are *static* [3, 5, 12, 15, 16]. Static load balanced join algorithms balance the join load at each node when the tasks are allocated to the nodes. The basic idea is to *detect* and *remedy* any load imbalance prior to join computation. Once the join computation begins, tasks will not migrate from one node to another. The exceptions to these trends are [9, 13].

In this paper, we revisit the problem of balancing the load for a join operation in shared-nothing systems. In particular, we proposed two dynamic algorithms. Dynamic algorithms distinguish themselves from the static ones by redistributing tasks, which may be partially executed, from one node to another. Here, load imbalance is detected and remedied during join computation. Whenever a node completes the processing of its allocated tasks, the idle node will "steal" some load from a busy node. The performance of the proposed algorithms is compared with two other static load-balanced join methods. The classic GRACE algorithm is used as a reference to measure the performance gain due to load balancing.

Several observations motivated this work. First, few study has been done on dynamically balancing the join load across the nodes in shared-nothing environment. Second, studies on load balancing join algorithms have shown that no single join algorithm is superior in all cases [3, 5]. When the data skew is low, GRACE hash join algorithm [2] that did not attempt to balance the load performs well enough. On the other hand, static load balancing join algorithms (such as adaptive load balancing join algorithm (ABJ) and extended adaptive load balancing join algorithm (ABJ^+)) are superior with high data skew. What is needed then is an algorithm that is robust to data skew. In particular, we want to keep the advantages of GRACE, which is its simplicity and good performance at low skew (since "the skew conditions are typically mild for many applications" [5]), and avoid its poor performance at high skew. Third, dynamism is important especially when statistics are inaccurate. Fourth, the availability of increasingly high-speed (and low-cost) interconnection network not only facilitates load-balancing but have made dynamic load-balancing techniques promising in handling load-imbalance.

The rest of the paper is organized as follows. In the next section, we review some related work. The load-balanced join algorithms studied in this work are presented in Section 3. In Section 4, we develop an analytical model that is used to compute the elapsed time for each of the join algorithms. The results of a performance study are presented in Section 5. Finally, we conclude in Section 6.

---

[1] The term "task" here refers to the join of two corresponding buckets from the two relations.

2 Related Work

Wolf, Dias and Yu propose two algorithms, a sort-based join [15] and a hash-based join [16], to handle data skew. Both algorithms preprocess the relations to collect statistics that are used in a *scheduling phase* that performed the following: The relations are logically split into a set of tasks. The estimated execution time of each task is computed. The tasks are then assigned to nodes to minimize the overall completion time of the join. If the allocation is not satisfactory, that is the completion time is not within some tolerance limits, the existing set of tasks are further split (logically) and the process repeats. The algorithm is heavily dependent on the estimation of the execution cost of the tasks. With poor estimation of these costs, the algorithm may not perform well. Moreover, it is shown to perform poorly for low data skew because ofthe overhead to preprocess the relations.

Besides using the estimated execution time as the criterion for allocation of tasks [15, 16], the load may also be balanced across the nodes based on bucket sizes [5, ing