On Parallel Hash-Join Processing with Skewed Data

6W－7

Lilian Harada
Fujitsu Laboratories

Masaru Kitsuregawa
University of Tokyo

1. Introduction

When data are uniformly distributed, parallel hash-join algorithm scales up well. However, the presence of data skew can cause load imbalances among the processors, significantly deteriorating its performance. Within the last years, there has been a growing interest in addressing the problem of join product skew, where the different join selectivities on processors lead to an imbalance on the number of output tuples. Here we present a dynamic skew handling algorithm which detects and rebalances unexpected join product skews at run-time in a shared-nothing database environment.

2. Our Dynamic Skew Handling Algorithm

2.1 Execution Phases

As the previous join product skew handling algorithms [DNSS92], [WDYT91], our algorithm is composed of four phases. In the Scan or Sampling Phase, statistics of fine partitions of the relations are obtained by scanning or sampling the operand relations. Then, in the Scheduling Phase, based on these statistics, the partitions join execution times are estimated and used to schedule the partitions to the processors so that the join execution times are almost equal for all processors. According to this scheduling, in the Redistribution Phase, data are exchanged between processors to assembly their partitions. Finally, when all partitions are assembled and locally stored at the processors, in the Join Phase each processor reads its local pair of partitions and joins it on its own site.

In our algorithm, during the Join Phase, the average processing rate of an initial portion of each partition is measured and compared to the partition's processing rate that was statically predicted and used in the Scheduling Phase. If their difference is significantly large to result in a disastrous performance degradation, the processor is detected as overloaded and thus, a workload compensation strategy is dynamically invoked. Otherwise, the processing continues as scheduled initially. In the case an overloaded processor is detected, the migration of some load from the overloaded processor to the non-overloaded processors is invoked. In this case, based on the measured partition's average processing rate, the amount of overload caused by the partition's unpredicted join product skew is calculated. Then, this overload is equally partitioned among the processors so that the relative execution times of the processors are maintained as estimated in the Scheduling Phase and thus, the processors' total execution times are re-equalized.

2.2 Overload Migration

The load migration from the overloaded processor to the non-overloaded processors is done in two different ways, depending on the non-overloaded processors' status.

(1) *Result Redistribution*: is applied when the non-overloaded processor is still processing its local pair of partitions. In this case, the memory of the non-overloaded processor is filled by its hash table and is not available to load any portion of the build or probe partitions of the overloaded processor. Therefore, the overloaded processor sends a portion of the result matching tuples to the non-overloaded processor, distributing the heavy task of writing the matching tuples among the processors. The non-overloaded processor writes the results received from the overloaded processor in its disk, besides concurrently continuing its own join processing.

(2) *Processing Task Migration:* is applied when the non-overloaded processor has already finished the processing of its local pair of partitions. In this case, the memory of the non-overloaded processor is free to load some portion of the build or probe partitions from the overloaded processor, so that a portion of matching tuples can be processed and produced locally on the non-overloaded processor. This approach is efficient when the join selectivity is much large, since from only some few pages of the build and probe partitions received from the overloaded processor, the non-overloaded processor can generate a large number of output pages locally. This can alleviate physical resource limitations of the CPU and network bandwidth.
There are two types of processing task migration: the *hash table migration* and the *probe migration*. In the hash table migration, the hash table in the overloaded processor is partitioned between the non-overloaded processors, and each portion of the hash table is migrated from the overloaded processor to the non-overloaded processors; all the remaining pages of the probe partition in the overloaded processor are broadcast to the non-overloaded processors. On

the other hand, in the probe migration, the hash table in the overloaded processor is simply replicated in the non-overloaded processors, while the remaining probe partition in the overloaded processor is partitioned between the non-overloaded processors. These two approaches are reciprocal: the hash table migration restricts the hash table size, while the probe partition is broadcast to the non-overloaded processors; on the other hand, the probe migration restricts the data flow of the probe partition, while the hash table is replicated in the non-overloaded processors.

3. Evaluation Results

In order to evaluate our dynamic join product skew algorithm, we made a series of experiments by varying the system and workload parameters. In the following we present only some representative simulation results, whose parameter values are summarized in Table 1. Concerning the skewed relations, we choose the skew model used in [DNSS92]. In this skew model, for a relation of size |R|, in each attribute the value 1 appears in some fixed number of tuples, while the remaining tuples contain values uniformly distributed between 2 and |R|. For example, the x10 attribute has the value 1 appearing in exactly ten tuples, while the remaining |R|-10 tuples contain values between 2 and |R|.

In Table 2 we present some simulation results for different levels of skew in both join relations, yet keeping the result size constant for each table. The results compare the performance of our dynamic join product skew handling algorithm with the static algorithm VP-RR introduced in [DNSS92]. The tables show that our approach outperforms VP-RR in all cases. In the presence of data skew, VP-RR performs poorly when some of their estimations largely differ from the actual values. However, our dynamic join product skew handling algorithm shows to successfully detect these unpredicted join product skews and re-balance the overload at run-time, resulting in large performance improvement.

read/write a disk page	20 msec
hash tuple	3 μsec
send message	5 msec
probe hash table	6 μsec
receive message	5 msec
join output tuple	40 μsec
check one processor overload	5 msec

(a) System Characteristics

cardinality of R	500,000 tuples
cardinality of S	500,000 tuples
tuple length of R	208 Bytes
tuple length of S	208 Bytes
tuple length of result (R,S)	416 Bytes
memory capacity	8 MBytes
number of nodes	30
page size	8 KBytes
message length	8 KBytes

(b) Workload Characteristics

Table 1. Parameters

time (s)	x10K,x10	x1K,x100	x100,x1K
VP-RR	39.9	63.5	139.9
Our Approach	38.9	39.3	40.4

(a) Result Size = 600 Ktuples

time (s)	x100K,x10	x10K,x100	x1K,x1K	x100,x10K
VP-RR	68.8	92.4	325.9	1087.0
Our Approach	66.7	71.7	72.0	71.8

(b) Result Size = 1.5 Mtuples

Table 2. Execution Times

4. Conclusion

In this paper we presented a dynamic skew handling algorithm for parallel hash-joins which re-balances the load during the Join Phase, when unpredicted join product skews are detected at run-time. Because of lack of space, here we only briefly introduced some basic ideas of our algorithm, without details about the overload detection, overload estimation, conditions for ideal migration, and the migration methods. We are now implementing this dynamic join product skew handling algorithm in our prototype parallel database system, whose results we intend to present elsewhere soon.

References

[DNSS92] D. DeWitt, J. F. Naughton, D. A. Schneider and S. Seshadri, "Practical Skew Handling in Parallel Joins", Proc. 18th. Int. Conf. VLDB, pp.27-40, 1992

[WDYT91] J. L. Wolf, D. M. Dias, P. S. Yu and J. Turek, "An Effective Algorithm for Parallelizing Hash Joins in the Presence of Data Skew", Proc. 7th. Int. Conf. DE, pp.200-209, 1991