Run Time Data Skew Handling on Parallel Hash Join

Yumiko Tsuchiya

School of Information Science,
Japan Advanced Institute of Science and Technology

February 14, 1997

Keywords: parallel database, hash join, join product skew, load balancing.

Research Background Today, parallel database systems are actively researched, and a lot of products are in circulation. Data skew handling is one of open problems on parallel database systems. That includes data skew handling on parallel hash join algorithms. The parallel hash join algorithms divide both join relations into disjoint partitions, called buckets, using a hash function. If attribute values are uniformly distributed, the algorithms can give each processing node even work loads, and shows good scaleup and speedup properties. But if many attribute values have a certain hash value, the performance suffers badly degradation, because of processing that high skewed bucket. In this paper, I describe parallel hash join algorithms and experiment results of data skew effect on that algorithm.

Existing Skew Handling Method For solving this load inbalanceing problem, many methods for treating with data skew have been proposed. At first, redistribution skew handling was concerned. The redistribution skew is inbalance in the number of tuples that is allocated to each processing node. The redistribution skew handling algorithms divide a relation into many buckets using a fine hash function, collect information of bucket size, and determine the optimal bucket allocation. These algorithms show better performance than conventional parallel hash join algorithms in simulation results using zipf-like data skew model, when join attribute values are non-uniformly distributed. In this paper, I give an overview of redistribution skew handling algorithms that were proposed in [KIT90] and [HUA95], and experiment results of difference of three bucket allocation strategys on processing time.

Even if these algorithms can handle the redistribution skew to make nearly even tuples be allocated to each processing nodes, load may be inbalanced due to difference of join

selectivity in each bucket. That inbalance is called join-product skew. If join product skew occur, quite many match tuples are produced by a certain high skewed bucket join. Thus processing node that allocated high skewed bucket incurs heavy load by writing result tuples into disks. To handle join-product skew, first, static join-product-skew handling algorithms were proposed. At first the algorithms scan or sample both join relations and gather partition statistics. Then they estimate join execution time, and assign tasks to optimal processors using the estimated partition join execution times. In this paper, I brief this static estimation method, and evaluate its estimation performance by using that method to estimate some non-uniform distribution models.

However the static join-product-skew handling algorithms depend on their static join execution time estimation. Thus if they fail in static estimation or join attribute has a high skewed attribute value, the algorithms cannot obtain enough effect. For these undesirable cases, dynamic join-product-skew handling algorithms were proposed. Dynamic join-product-skew handling algorithm in [HAR95] monitors each bucket joins, compares their processing rate to the statically estimated rate, and detects unpredictable join product skew. If it is detected, the algorithm dynamically migrate the detected overload to other non-overloaded processors. Thus the load of each node is dynamically balanced. In this paper, I reconsider the algorithm in more detail and experiment results concerning overload detection and run-time re-estimation based on the algorithm.

Dynamic Join Product Skew Handling by Distributed Coordinator In [HAR95]'s dynamic join-product-skew handling algorithm, a coordinator monitors each bucket join, maintains the status of each processing nodes, and determines the optimal load migration strategys using these gathered information, resulting effective dynamic load balancing. However the number of processors will continue to increase for supporting larger databases. In that case, a lot of bucket join information and skew handling process are concentrate at a single coordinator. The coordinator's load to gather many information, manage timers for each nodes, calculate the join processing rate, in overload detection case, re-estimate overload may and determine the migration strategys surpass its ability for such environment. So I study the strategy to distribute the process for achieved dynamic skew handling so as to remedy that overconcentration.

In the following, the strategy distributes the coordinator process, locally detects the overload by each processor's own decision, and migrates this overload if necessary.

Overload Detection In each bucket process, the local coordinator process on each node counts the numbers of processed probing tuples and match tuples generated by these probing tuples, calculates the average number of generated tuples per one probing tuple, called the brow-up ratio, and compares the measured brow-up ratio to predicted one. When a miss estimation is detected, it means that the node processing this bucket join is overloaded. So in that case, the local coordinator process finds itself be overloaded.

Migrated Process To perform the overload migration, the overloaded node migrates tuples generated by own probing process to write them in the other nodes. As a

lot of match tuples is generated, these match tuples concentrated to a disk written by the overloaded node, resulting heavy load. It can reduce processing time of overloaded node and remedy skew of disk utilization.

Amount of Migrated Process Once the overloaded is detected in a bucket, the overloaded node migrate a portion of match tuples that are generated by the bucket join to all the others. The amount of migrated match tuples is $(N-1)/N$ of the number of match tuples generated by each probing tuple. Here, N is the number of processing nodes. Thus $1/N$ of the all writing tasks of the overloaded node are distributed to each of all processing node. But in the case where the number of match tuples generated by a probing tuple is smaller than the predicted brow-up ration, these match tuples are not migrated to other nodes for handling miss overload detection.

In such a way, I try to omit maintaining each processing nodes status, therefor employ the monotonous migration method which let non-overloaded nodes write match tuples generated in overloaded node to disk. In that case, it is difficult to employ the migration strategy using each processing node status.

For the studying performance of such dynamic load balancing method, I implemented the method and derive the performance results using nCUBE/2. In this paper, I show the experiment results that measured the number of written by each processor and processing time of each bucket join under using that distributed dynamic join-product-skew handling method. The results illustrate that the strategy has standard deviation 1/3 times as little as the conventional parallel hash join algorithm thus it has less dispersion. They indicate in the best case, the strategy can process a high skewed bucket join 0.67 times as fast as it without skew handling.

References

[DWT92] D.J. DeWitt, J.F. Naughton, D.A. Schneider, S. Seshadri "Practical Skew Handling in Parallel Joins" Proc. 18th VLDB Conf. 1992

[HUA95] K.A. Hua, C. Lee, C.M. Hua "Dynamic Load Balancing in Multicomputer Database Systems Using Partition Tuning" IEEE Trans. Knowledge and Data Engineering. vol. 7, No. 6, Dec. 1995

[KIT90] M. Kitsuregawa, Y. Ogawa "Bucket Spreading Parallel Hash: A New, Robust, Parallel Hash Join Method for Data Skew in the Super Database Computer" Proc. 16th Conf. VLDB 1990

[HAR95] L. Harada, M. Kitsuregawa "Dynamic Join Product Skew Handling for Hash-Joins in Shared-Noting Database Systems" Proc. 4th Inter. Conf. on DASFAA'95, Apr. 1995

[WOL91] J.L. Wolf, D.M. Dias, P.S. Yu, J. Turek "An Effective Algorithm for Parallelizing Hash Joins in the Presence of Data Skew" Proc. 7th Int. Conf. DE 1991