Extreme Skew Handling in Right-Deep Multi-Joins

1R－7

Stephen Davis Masaru Kitsuregawa
University of Tokyo, Institute of Industrial Science

1 Introduction

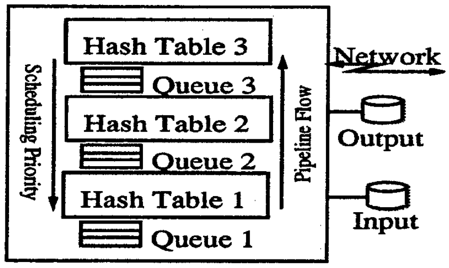
Execution of relational operations on a shared nothing system is susceptible to various forms of skew due to the partitioning of tuples amongst the processing nodes. This skew reduces the overall efficiency of the multi-join operation. In this paper, we present an algorithm for dynamically handling skew in a right-deep hash multi-join.

In the right-deep hash multi-join, a right-deep query execution plan is used to schedule the order in which the pair-wise hash joins should be performed. The right-deep query allows the multi-join to proceed in a pipelined manner. When this pipeline is partitioned amongst the processors, skew affects the flow of tuples through the processors, allowing some processors to finish reading all of the tuples on their local disks much faster than others. Once all of the disk tuples have been read, the processor will become periodically idle while it waits for probe tuples to arrive from another processor. As the skew in the disk read completion times increases, processor utilization efficiency decreases which leads to a longer response time for the multi-join.

Our algorithm makes use of a control processor, called the foreman, to make decisions about how hash lines should be migrated in order to balance the disk completion times, where a hash line consists of the set of build relation tuples of a particular pair-wise join which map to the same hash entry in the hash table. A hash line is a natural unit of migration, since it assures that all build relation tuples with the same attribute value are located on the same processor. Load balancing occurs in two phases, the first phase balances the pipeline flow for each processor, and the second phase balances the disk completion times.

2 Processing Model

All of the processors are assigned to work on all of the pair-wise joins of the multi-join. For each join, the build relation tuples are partitioned amongst the processors, and each of these partitions is called a stage fragment. Each processor has one stage fragment for each join (stage) in the right-deep query plan. Because each processor is working on all joins, a scheduling policy is required to determine which join a processor should work on at a particular point in time. A processor's stage fragments which are eligible to execute are the one's with probe tuple's in their input queues. Of the eligible fragments, the one selected is the one closest to the output end of the pipeline. This is to reduce the accumulation of buffers in the processor's stage fragment input queues. Thus later stage fragments have higher priority than earlier ones.



Processor $_i$

Because of this prioritization, as long as later stage fragments have buffers in their input queues, earlier stage fragments within that processor will receive no execution time. Thus, they are not able to consume the probe tuples received from other processors, and to avoid running out of buffers, flow control becomes necessary. If a later stage fragment takes longer to process a buffer of probe tuples than another stage fragment for the same stage, the probability that a new buffer will be received by one of the later stages on that processor increases, thus decreasing the likelihood that earlier stage fragments will receive processing time.

If pipeline flow is fairly uniform, then a processor will spend roughly the same percentage of time working on a particular stage as do the other processors. To obtain uniform flow, dynamic load balancing is required.

3 Pipeline Stage Balancing

The goal of the first load balancing phase is to even out the pipeline flow amongst the processors. This way, even if all of the processors are spending more time processing later stages, flow control is not needed since earlier stages are uniformly receiving less processing time.

The load balancing algorithm uses a control processor, called the foreman, to gather join processing statistics from each of the processors and to determine how hash lines should be migrated to evenly distribute the stage loads. Statistics gathering begins either when flow control is activated, or periodically based on the number of tuples processed. When to gather is determined by the join processors.

Once the foreman receives a request to begin statistics gathering, it sends a message to all processors requesting the statistics. The statistics gathered by the foreman are the number of probes performed, number of tuples compared against for matches of the join criterion, and the number of *results tuples* produced by each stage fragment.

3.1 Completion Time Estimation

Based on the statistics received, the foreman estimates how much time remains for processing a particular stage fragment. The assumption made by the foreman is that the distribution of probe tuples attributes already processed is representative of the distribution of the remaining probes. The remaining time estimates are computed on a stage by stage basis, beginning with the first stage in the pipeline.

For the first stage, the total number of tuples remaining to be probed is the number of unprocessed tuples from the disks, since selection is not being performed. This can be computed from the total number of probe tuples originally on the disks minus the number of tuples already probed at the first stage. The estimated remaining probe tuples multiplied by the percentage of total probes probed at the stage fragment is the estimated number of remaining probes to be performed by that stage fragment.

The estimated remaining time for a stage fragment is then computed by estimating the number of remaining comparisons and remaining result tuples to generate, multiplying these estimates by the time required to perform a comparison and to generate a result.

The remaining times for the stage fragments in the later stages is computed iteratively, using the estimated output of the previous stage as the number of tuples to be probed at the current stage.

3.2 Hash Line Migration

If the skew amongst the stage fragments of a stage is sufficient, hash lines will be migrated amongst the stage fragments to equalize the remaining times. The stage's stage fragments which send hash lines are those whose remaining times are sufficiently higher than the average remaining time, and the rest are receivers.

Once the remaining times are determined, the foreman requests the hash line statistics (tuples probed, hash line length, results generated) for all of the hash lines in the sender stage fragments. For each hash line, the estimated remaining processing time is determined. Based on the estimates, hash lines are assigned from senders to receivers within the same stage so that the time remaining for the stage is fairly uniformly distributed amongst the stage's stage fragments. These migrations maps are sent to all of the join processors, and the required hash lines are moved between the processors.

4 Stage to Disk Balancing

After stage balancing has been performed several times, each stage's stage fragments are fairly uniformly balanced, and the pipeline flow is smooth. When there are no stages requiring balancing, the disk completion balancing phase is entered.

When skew is present, flow control gets invoked very early, thus statistics are insufficient to properly balance everything, hence stage balancing is performed several times early on as a result of flow control being invoked. Once flow has been equalized sufficiently flow control is no longer necessary. After stage balancing has been completed, the rate at which each processor reads from disk will be fairly uniform, however the variance in the number of tuples remaining to be read will affect the overall response time. The emphasis of disk completion balancing is to reduce this variance, thus increasing the period of time in which all processors are fully utilized.

5 Disk Completion Balancing

Statistics gathering and estimation of the completion times are performed as in the stage balancing phase. However, instead of balancing stage times, overall processor completion times are balanced. The remaining time for a processor is the sum of the remaining times of each of its stage fragments plus the time spent forwarding tuples read from disk to the appropriate stage fragment. The senders are the processors whose remaining disk tuples are significantly larger than average and whose remaining processing time is greater than the average. The stage chosen to send from is the one with the largest remaining time and the amount to send is based on the difference between the processor time and the average.

6 Experiments

The load balancing algorithm was evaluated through simulation, with the actual joins being performed. Each tuple is 256 bytes. The join attributes are four byte integers. A compare takes 3 time units (TUs), result generation 256 TUs, send and receives through the network take a minimum of 1024 TUs, as do reads and writes from disk. The relation attributes are Zipf distributed. The number of tuples generated by each stage fragment in the first and third stages are nearly uniform. In the second stage, one processor generates 40% of the result tuples. Each of the build relations has 240,000 tuples, and the initial base probe relation has 960,000 tuples. The tuples of the build relations are nearly uniformly distributed amongst the processors. The execution times are presented in millions of TUs, measured from the start of the probe phase.

Unbalanced time	Balanced time
443.6	280.2

There were seven load balances (5 stage [3 flow control, 2 periodic], 2 disk completion) moving approximately 2.4MB of build tuples between processors.

7 Conclusion

Evening out pipeline flow and reducing the disk read completion times improves the multi-join processing efficiency, reducing the overall response time.

References

[1] D. Schneider and D.J. DeWitt. "Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines." Proc. of the 16th Int'l Conf. on Very Large Data Bases, August 1990.