CS 186 Spring 2024

Introduction to Database Systems
Lakshya Jain

DIS 11

1 Parallel Query Processing

1. What is the difference between inter- and intra- query parallelism?

Inter-query parallelism operates between multiple queries, rather than within a single query, whereas intra-query parallelism operates within a single query (parallelism of the operators that make up the query).

2. What are the advantages and disadvantages of organizing data by keys?

Advantages: because data is organized by keys, search and update operations (which require searching on the key) can be done more efficiently, since we have some sense of where the data must be (if it exists).

Disadvantages: we must maintain the organization, which adds overhead to insertions and updates.

3. All of the data for a relation with N pages starts on one machine, and we would like to partition the data onto M machines. Assume that the size of each page is S (in KB).

   a. How much data (in KB) would be sent over the network to partition the data through each of the following: range, hash, and round-robin partitioning? Assume we use uniform hash functions and are able to construct ranges that have the same number of values in them.

   The amount of data sent over the network for all three kinds of partitioning will be the same since we have a uniform spread of the data across the ranges and a uniform hash function. In this average case, each machine would get 1/M of the data. Since we don't count the amount of data on our current machine, we would need to send data to the other M-1 machines. The total amount of data sent over the network is $S * N * \frac{M-1}{M}$ KB.

   b. If there are no assumptions about hash functions or data ranges, what is the best and worst case in terms of network I/O cost for the three partitioning schemes?

   For round-robin partitioning, the best and worst case is still $S * N * \frac{M-1}{M}$ KB, since the data will still be divided evenly. For range and hash partitioning, the best case will be 0 KB, if all the data stays on the current machine. The worst case will be $S * N$ KB, in the scenario where the current machine retains no data and sends all of it to the other machines.

4. Assume for parts (a) and (b) that we have m=3 machines with B=5 buffer pages each, along with N=63 pages of data that don't contain duplicates.

a. In the best case, what is the number of passes needed to sort the data?

We use range partitioning for parallel sorting, so the first pass through the data will be for range partitioning the data among the 3 machines. In the best case, we can partition the data pages evenly among the 3 machines, and each machine will contain 21 pages.

Then, we execute the external sorting algorithm on each machine.

In Pass 1 of external sorting, we produce 4 sorted runs with 5 pages and 1 sorted run with 1 page on each machine.

Then in Pass 2, reserving 4 input buffers and 1 output buffers, we merge these runs 4 at a time, producing 2 sorted runs.

Finally, in pass 3, we are able to merge together these 2 sorted runs, finishing off the external sorting process.

Thus, the sorting process takes 4 passes overall (1 pass for partitioning the data, and 3 passes for executing external sorting on each machine).

In general, the number of passes needed to sort the data in the best case is: (number of passes to partition the data) + (number of passes to sort each partition)

$$1 + \lceil 1 + log_{B-1}\lceil \frac{N}{mB} \rceil \rceil$$

b. What is the number of passes needed to hash the data? Examine the best case scenario, where the data will be uniformly distributed under the given hash functions.

The total number of passes is: (the number of passes to partition the data) + (the number of passes to hash each partition)

We use hash partitioning for parallel hashing, so the first pass through the data will be for hash partitioning the data among the 3 machines. The data is uniformly distributed by the hash function we used to partition the data, so each machine ends up with 21 pages of data.

In the first partitioning pass, reserving 1 input buffer and 4 output buffers, each machine will partition 21 data pages into 4 partitions of 21 / 4 = 5.25 –> 6 pages each.

In the second partitioning pass, we split each partition of 6 pages each into 4 partitions of 6 / 4 = 1.5 –> 2 pages each (ending up with 16 partitions of 2 pages each total).

After that, the partitions are small enough to fit in memory, so we can proceed with the conquer phase - we make one more pass through the data to read in the pages in each partition, construct an in-memory hash table, and write that back to disk.

Thus, the process will take 4 passes total.

c. Assume that relation R has R pages of data, and relation S has S pages of data. If we have m machines with B buffer pages each, what is the number of passes in order to perform sort merge join (in terms of R, S, m, and B) in the best case? Consider reading over either relation to be a pass.

(1 pass to partition each of the two tables across machines) + (the number of passes needed to sort R) + (the number of passes to sort S) + (1 final merge sort pass, going through both tables)

$$2 + \lceil 1 + log_{B-1}\lceil \frac{R}{mB} \rceil \rceil + \lceil 1 + log_{B-1}\lceil \frac{S}{mB} \rceil \rceil + 2$$

d. Can you use pipeline parallelism to implement the join from part c?

No, the sorting pass must complete before the merge pass can begin.

5. Relation R has 10,000 pages, round-robin partitioned across 4 machines (M1, M2, M3, M4). Relation S has 10 pages, all of which are only stored on M1. We want to join R and S on the condition R.col = C.col. Assume the size of each page is 1 KB.

a. What type of join would be best in this scenario, and why?

Broadcast join, because it is cheaper to send relation S to every machine rather than to partition R based on col.

b. How many KB of data must be sent over the network to join R and S?

The amount of data sent over the network is the amount of data required to send all pages of S to every machine that does not have it (M2, M3, and M4): 3 * 10 = 30 KB

c. Would the amount of data sent over the network change if R was hash partitioned among the 4 machines rather than round-robin partitioned? What if R was instead range partitioned? Assume each machine contains at least 1 tuple of R after partitioning.

If we were to use broadcast join, the network cost would remain the same regardless of whether R was hash partitioned or range partitioned, assuming there are tuples of R on each of the 4 machines.

However, if R was hash partitioned across the 4 machines, we might be able to get a lower network cost by using parallel Grace Hash Join. Note that we would need to keep track of the hash function used to partition R among the M machines, so that we can use the same hash function to hash partition S. Note that this would be an asymmetric shuffle.

Similarly, if R was range partitioned across the 4 machines, we might be able to get a lower network cost by using parallel Sort Merge Join. Note that we would need to keep track of the ranges used to partition S among the m machines, so that we can use the same ranges to range partition S. Note that this would be an asymmetric shuffle.