CHAPTER 22

Parallel and Distributed Query Processing

Practice Exercises

22.1 What form of parallelism (interquery, interoperation, or intraoperation) is likely to be the most important for each of the following tasks?

a. Increasing the throughput of a system with many small queries

b. Increasing the throughput of a system with a few large queries when the number of disks and processors is large

22.2 Describe how partial aggregation can be implemented for the **count** and **avg** aggregate functions to reduce data transfer.

22.3 With pipelined parallelism, it is often a good idea to perform several operations in a pipeline on a single processor, even when many processors are available.

a. Explain why.

b. Would the arguments you advanced in part *a* hold if the machine has a shared-memory architecture? Explain why or why not.

c. Would the arguments in part *a* hold with independent parallelism? (That is, are there cases where, even if the operations are not pipelined and there are many processors available, it is still a good idea to perform several operations on the same processor?)

22.4 Consider join processing using symmetric fragment and replicate with range partitioning. How can you optimize the evaluation if the join condition is of the form $|r.A - s.B| \leq k$, where k is a small constant? Here, $|x|$ denotes the absolute value of x. A join with such a join condition is called a **band join**.

22.5 Suppose relation r is stored partitioned and indexed on A, and s is stored partitioned and indexed on B. Consider the query:

$$_{r.C}\gamma_{\text{count}(s.D)}(\ (\sigma_{A>5}(r)) \bowtie_{r.B=s.B} s\ )$$

a. Give a parallel query plan using the exchange operator, for computing the subtree of the query involving only the select and join operators.

b. Now extend the above to compute the aggregate. Make sure to use preaggregation to minimize the data transfer.

c. Skew during aggregation is a serious problem. Explain how preaggregation as above can also significantly reduce the effect of skew during aggregation.

22.6 Suppose relation r is stored partitioned and indexed on A, and s is stored partitioned and indexed on B. Consider the join $r \bowtie_{r.B=s.B} s$. Suppose s is relatively small, but not small enough to make asymmetric fragment-and-replicate join the best choice, and r is large, with most r tuples not matching any s tuple. A hash-join can be performed but with a semijoin filter used to reduce the data transfer. Explain how semijoin filtering using Bloom filters would work in this parallel join setting.

22.7 Suppose you want to compute $r \rtimes\!\bowtie_{r.A=s.A} s$.

a. Suppose s is a small relation, while r is stored partitioned on $r.B$. Give an efficient parallel algorithm for computing the left outer join.

b. Now suppose that r is a small relation, and s is a large relation, stored partitioned on attribute $s.B$. Give an efficient parallel algorithm for computing the above left outer join.

22.8 Suppose you want to compute $_{A,B}\gamma_{sum(C)}$ on a relation s which is stored partitioned on $s.B$. Explain how you would do it efficiently, minimizing/avoiding repartitioning, if the number of distinct $s.B$ values is large, and the distribution of number of tuples with each $s.B$ value is relatively uniform.

22.9 MapReduce implementations provide fault tolerance, where you can reexecute only failed mappers or reducers. By default, a partitioned parallel join execution would have to be rerun completely in case of even one node failure. It is possible to modify a parallel partitioned join execution to add fault tolerance in a manner similar to MapReduce, so failure of a node does not require full reexecution of the query, but only actions related to that node. Explain what needs to be done at the time of partitioning at the sending node and receiving node to do this.

22.10 If a parallel data-store is used to store two relations r and s and we need to join r and s, it may be useful to maintain the join as a materialized view. What are

the benefits and overheads in terms of overall throughput, use of space, and response time to user queries?

22.11 Explain how each of the following join algorithms can be implemented using the MapReduce framework:

a. Broadcast join (also known as asymmetric fragment-and-replicate join).

b. Indexed nested loop join, where the inner relation is stored in a parallel data-store.

c. Partitioned join.