CHAPTER 18

Parallel Databases

Practice Exercises

18.1 In a range selection on a range-partitioned attribute, it is possible that only one disk may need to be accessed. Describe the benefits and drawbacks of this property.

Answer: If there are few tuples in the queried range, then each query can be processed quickly on a single disk. This allows parallel execution of queries with reduced overhead of initiating queries on multiple disks.

On the other hand, if there are many tuples in the queried range, each query takes a long time to execute as there is no parallelism within its execution. Also, some of the disks can become hot-spots, further increasing response time.

Hybrid range partitioning, in which small ranges (a few blocks each) are partitioned in a round-robin fashion, provides the benefits of range partitioning without its drawbacks.

18.2 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

Answer:

a. When there are many small queries, inter-query parallelism gives good throughput. Parallelizing each of these small queries would increase the initiation overhead, without any significant reduction in response time.

b. With a few large queries, intra-query parallelism is essential to get fast response times. Given that there are large number of processors and disks, only intra-operation parallelism can take advantage of the parallel hardware – for queries typically have few operations, but each one needs to process a large number of tuples.

18.3 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?)

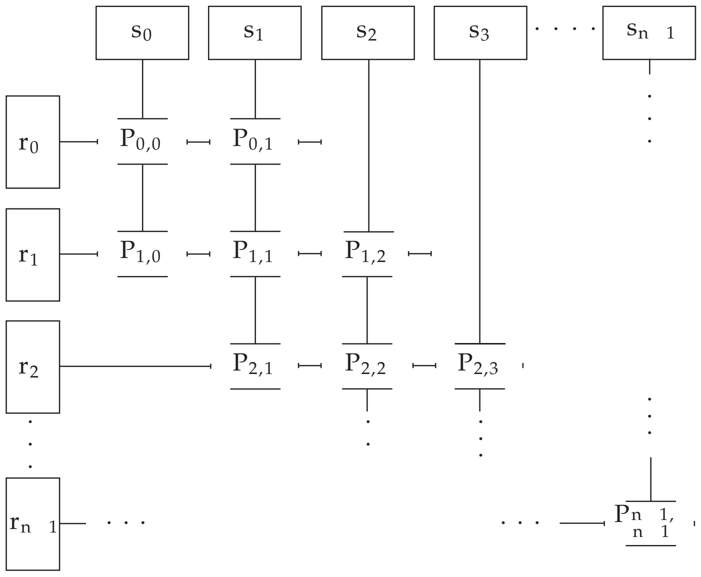
Answer:

a. The speed-up obtained by parallelizing the operations would be offset by the data transfer overhead, as each tuple produced by an operator would have to be transferred to its consumer, which is running on a different processor.

b. In a shared-memory architecture, transferring the tuples is very efficient. So the above argument does not hold to any significant degree.

c. Even if two operations are independent, it may be that they both supply their outputs to a common third operator. In that case, running all three on the same processor may be better than transferring tuples across processors.

18.4 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 $\mid r.A - s.B \mid \ \leq k$, where k is a small constant? Here, $\mid x \mid$ denotes the absolute value of x. A join with such a join condition is called a **band join**.
Answer: Relation r is partitioned into n partitions, $r_0, r_1, \ldots, r_{n-1}$, and s is also partitioned into n partitions, $s_0, s_1, \ldots, s_{n-1}$. The partitions are replicated and assigned to processors as shown below.



Each fragment is replicated on 3 processors only, unlike in the general case where it is replicated on n processors. The number of processors required is now approximately $3n$, instead of n^2 in the general case. Therefore given the same number of processors, we can partition the relations into more fragments with this optimization, thus making each local join faster.

18.5 Recall that histograms are used for constructing load-balanced range partitions.

a. Suppose you have a histogram where values are between 1 and 100, and are partitioned into 10 ranges, 1–10, 11–20, . . . , 91–100, with frequencies 15, 5, 20, 10, 10, 5, 5, 20, 5, and 5, respectively. Give a load-balanced range partitioning function to divide the values into 5 partitions.

b. Write an algorithm for computing a balanced range partition with p partitions, given a histogram of frequency distributions containing n ranges.

Answer:

a. A partitioning vector which gives 5 partitions with 20 tuples in each partition is: $[21, 31, 51, 76]$. The 5 partitions obtained are $1-20, 21-30, 31-50, 51-75$ and $76-100$. The assumption made in arriving at this partitioning vector is that within a histogram range, each value is equally likely.

b. Let the histogram ranges be called $h_1, h_2, \ldots, h_h$, and the partitions
$p_1, p_2, \ldots, p_p$. Let the frequencies of the histogram ranges be

$n_1, n_2, \ldots, n_h$. Each partition should contain N/p tuples, where $N = \Sigma_{i=1}^{h} n_i$.
To construct the load balanced partitioning vector, we need to determine the value of the k_1^{th} tuple, the value of the k_2^{th} tuple and so on, where $k_1 = N/p, k_2 = 2N/p$ etc, until k_{p-1}. The partitioning vector will then be $[k_1, k_2, \ldots, k_{p-1}]$. The value of the k_i^{th} tuple is determined as follows. First determine the histogram range h_j in which it falls. Assuming all values in a range are equally likely, the k_i^{th} value will be

$$s_j + (e_j - s_j) * \frac{k_{ij}}{n_j}$$

where

s_j	:	first value in h_j
e_j	:	last value in h_j
k_{ij}	:	$k_i - \Sigma_{l=1}^{j-1} n_l$

18.6 Large-scale parallel database systems store an extra copy of each data item on disks attached to a different processor, to avoid loss of data if one of the processors fails.

a. Instead of keeping the extra copy of data items from a processor at a single backup processor, it is a good idea to partition the copies of the data items of a processor across multiple processors. Explain why.

b. Explain how virtual-processor partitioning can be used to efficiently implement the partitioning of the copies as described above.

c. What are the benefits and drawbacks of using RAID storage instead of storing an extra copy of each data item?

Answer: FILL

18.7 Suppose we wish to index a large relation that is partitioned. Can the idea of partitioning (including virtual processor partitioning) be applied to indices? Explain your answer, considering the following two cases (assuming for simplicity that partitioning as well as indexing are on single attributes):

a. Where the index is on the partitioning attribute of the relation.

b. Where the index is on an attribute other than the partitioning attribute of the relation.

Answer: FILL

18.8 Suppose a well-balanced range-partitioning vector had been chosen for a relation, but the relation is subsequently updated, making the partitioning unbalanced. Even if virtual-processor partitioning is used, a particular virtual processor may end up with a very large number of tuples after the update, and repartitioning would then be required.

a. Suppose a virtual processor has a significant excess of tuples (say, twice the average). Explain how repartitioning can be done by splitting the partition, thereby increasing the number of virtual processors.

b. If, instead of round-robin allocation of virtual processors, virtual partitions can be allocated to processors in an arbitrary fashion, with a mapping table tracking the allocation. If a particular node has excess load (compared to the others), explain how load can be balanced.

c. Assuming there are no updates, does query processing have to be stopped while repartitioning, or reallocation of virtual processors, is carried out? Explain your answer.

Answer: FILL