CS 186 - Fall 2020
Guerrilla Section 5
Parallel Query Processing and Database Design

Sunday, November 15, 2020

1 Types of Parallelism

For each of the following scenarios, state whether it is an example of:

- Inter-query parallelism
- Intra-query, inter-operator parallelism
- Intra-query, intra-operator parallelism
- No parallelism

1. A query with a selection, followed by a projection, followed by a join, runs on a single machine with one thread.

   Answer: No parallelism. It might look like a pipeline, but at any given point in time there is only one thing happening, since there is only one thread.

2. Same as before, but there is a second machine and a second query, running independently of the first machine and the first query.

   Answer: Inter-query parallelism.

3. A query with a selection, followed by a projection, runs on a single machine with multiple threads; one thread is given to the selection and one thread is given to the projection.

   Answer: Intra-query, inter-operator parallelism.

4. We have a single machine, and it runs recursive hash partitioning (for external hashing) with one thread.

   Answer: No parallelism, because there is only one machine and one thread. Don't confuse this with parallel hashing!

5. We have a multi-machine database, and we are running a join over it. For the join, we are running parallel sort-merge join.

   Answer: Intra-query, intra-operator parallelism. We have a single query and a single operator, but that single operator is going to do multiple things at the same time (across different machines).

2 Partitioning for Parallelism

1. Suppose we have a table of size 50,000 KB, and our database has 10 machines. Each machine has 100 pages of buffer, and a page is 4 KB.

   We would like to perform parallel sorting on this table, so first, we perfectly range partition the data. Then on each machine, we run standard external sorting.

   How many passes does this external sort on each machine take?

   Answer: **2 passes**.
   After range partitioning, each table will have 5,000 KB of data, or 1,250 pages. With 100 pages of buffer, this will take 2 passes to sort.

2. Suppose we were doing parallel hash join. The first step is to partition the data across the machines, and we usually use hash partitioning to do this.

   Would range partitioning also work? What about round-robin partitioning?

   Answer: Range partitioning also works, because items with the same key still end up on the same machine as required. Round-robin partitioning does not do that, so it does not work.

3. Suppose we have a table of 1200 rows, perfectly range-partitioned across 3 machines in order.

   We just bought a 4th machine for our database, and we want to run parallel sorting using all 4 machines.

   The first step in parallel sorting is to repartition the data across all 4 machines, using range partitioning. (The new machine will get the last range.)

   For each of the first 3 machines, how many rows will it send across the network during the repartitioning? (You can assume the new ranges are also perfectly uniform.)

   Answer: **100 rows, 200 rows, and 300 rows.**
   The original partitions were 3 ranges of 400 rows each; the new 4 ranges will have 300 rows each.
   The first machine held the first 400 rows originally, and now only needs to hold the first 300. It will send the remaining 100 rows over the network (to machine 2).
   The second machine held rows 401-800 initially, but now needs to hold rows 301-600. It will send rows 601-800 (200 rows) to machine 3.
   The third machine held rows 801-1200 initially, and similarly needs to send rows 901-1200 (300 rows) to machine 4.

3 Parallel Query Processing

Suppose we have 4 machines, each with 10 buffer pages. Machine 1 has a `Students` table which consists of 100 pages. Each page is 1 KB, and it takes 1 second to send 1 KB of data across the network to another machine.

1. How long would it take to send the data over the network after we uniformly range partition the 100 pages? Assume that we can send data to multiple machines at the same time.
   Answer: **25 seconds.**
   After we uniformly partition our data, Machine 1 will send 25 pages to Machines 2, 3, and 4. It will take 25 seconds to finish sending these pages to each machine if we send the pages to each machine at the same time.

2. Next, imagine that there is another table, `Classes`, which is 10 pages. Using just one machine, how long would a BNLJ take if each disk access (read or write) takes 0.5 seconds?
   Answer: **105 seconds.**
   BNLJ will require 10 + ceil(10/8) * 100 = 210 I/Os, which will take 105 seconds.

3. Now assume that the `Students` table has already been uniformly range partitioned across the four machines, but `Classes` is only on Machine 1. How long would a broadcast join take if we perform BNLJ on each machine? Do not worry about the cost of combining the output of the machines.
   Answer: **40 seconds.**
   First, we must broadcast the `Classes` table to each machine, which will take 10 seconds (since we send the table to each machine at the same time). Next, we will perform BNLJ on each machine, which requires 10 + ceil(10/8) * 25 = 60 I/Os, or 30 seconds. In total, the time required to send the data over the network and perform the join will be 40 seconds.

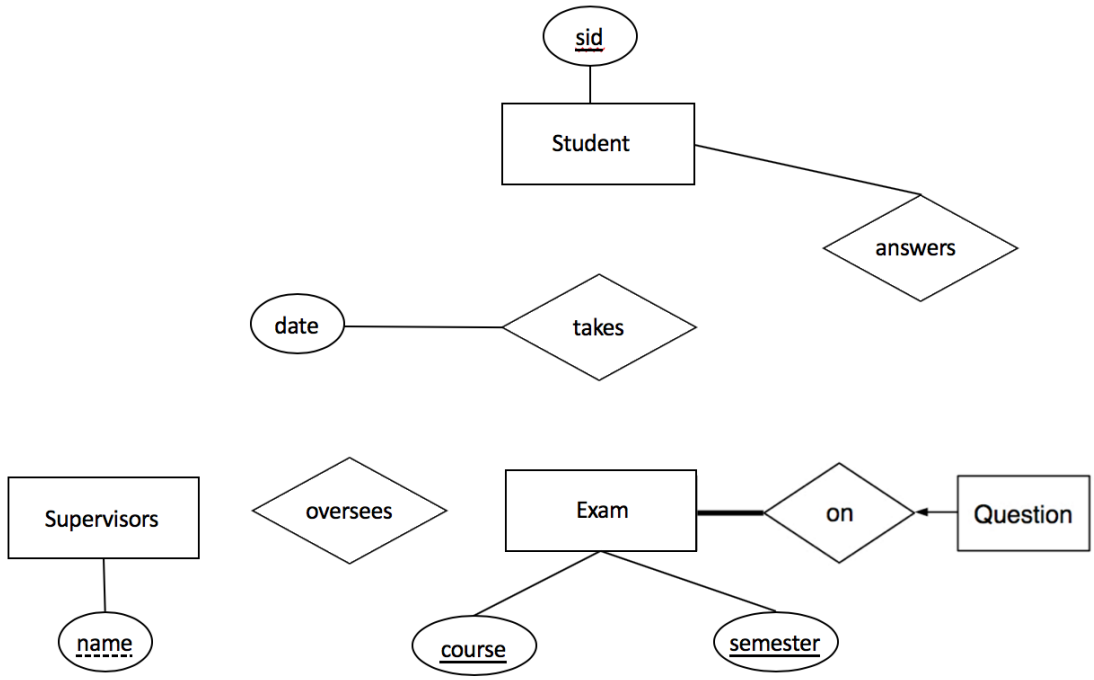
4. Which algorithm performs better?
   Answer: Broadcasting the `Classes` table and performing a parallel BNLJ runs faster than just using one machine, even with the additional time required to send the table over the network.

5. Knowing that the `Students` table was range partitioned, how can we improve the performance of the join even further?
   Since we know the `Students` table was range partitioned, we can also range partition the `Classes` table on the same column and only send the corresponding partitions to each machine. This should lower the network cost and decrease the number of disk I/Os.

4 Database Design

As chancellor of UC Berkeley, you are tasked with designing a new final exam scheduling system to make it easier on students. Using the following assumptions, fill in the ER diagram we give you.

- A student may take any number of exams, and every exam is taken by at least one student.
- An exam is uniquely identified by the combination of a course and a semester.
- Every exam has at least one supervisor. A supervisor oversees exactly one exam.
- There is at least one question on every exam, and a question appears on at most one exam.
- A question on an exam may be answered by any number of students, and a student may answer any number of questions on an exam.



1. What type of edge should be drawn between the **Supervisors** entity and the **oversees** relationship set?

   Bold Arrow - Each supervisor must oversees exactly one exam. This means that a supervisor must have a key constraint and participation constraint on their relationship with **oversees**.

2. What type of edge should be drawn between the **Exam** entity and the **oversees** relationship set?

   Bold Line - At least one supervisor oversees the exam. This is a participation constraint.

3. What type of edge should be drawn between the **Student** entity and the **takes** relationship set?

   Thin Line - Each student may take 0 or 1 or more exams on one day.

4. What type of edge should be drawn between the **Exam** entity and the **takes** relationship set?

   Bold Line - Student takes at least one exam in total. It's participation constraint.

5. What type of edge should be drawn between the **Questions** entity and the **answers** relationship set?

   Thin Line - Each **student** may answers 0, 1 or more **questions**.

6. Consider the attribute set R = ABCDEF and the functional dependency set F = {BE → C, B → F, D → F, AEF → B, A → E}. Which of the following are candidate keys of R? Mark all that apply

   (A) ACD

   (B) AD

   (C) FC

   (D) BF

   In computing attribute closure of a key K, we repeatedly process the set of functional dependencies F and add on attributes to the closure of K until there is nothing more that can be added. For ACD, we see that this leads first to ACDEF on the first processing of F and ABCDEF upon the second time we go through the loop. This means ACD is a superkey; however, for it to be a candidate key, we should check and make sure none of its subsets are superkeys for the table. Processing AD, however, gets us ADEF on the first iteration of the loop, ABDEF on the second, and ABCDEF on the third.

   FC and BF are not superkeys (and therefore not candidate keys): both are their own attribute closures.

7. Given Attribute Set R = ABCDEFGH and functional dependencies set F = {CE → GH, F → G, B → CEF, H → G}. What relations are included in the final decomposition when decomposing R into BCNF in the order of functional dependencies set F?

   - CE → GH violates BCNF, decompose into ABCDEF CEGH.
   - F → G No relation contains FG, skip.
   - B → CEF violates BCNF, decompose into ABD, BCEF, and CEGH.
   - H → G violates BCNF, decompose into ABD, BCEF, CEH, GH.

   Final relations are ABD, BCEF, CEH, GH.

8. True or False: The decomposition of attribute set R = ABCDEF, given the functional dependency set F = {B → D, E → F, D → E, D → B, F → BD}, into ABDE, BCDF is lossless.

   False, it is lossy. ABDE ∩ BCDF = BD, BD → BDEF, which is not a superset of either ABDE or BCDF.