ACS Theory Assignment 3

Rikke Tophøj Heinemann (DTZ190)
Marie Elkjær Rødsgaard (DCK495)
Rasmus Pallisgaard (DWP992)

January 23, 2024

1 Question 1: Data Processing I

We are given the two following tables

```
Orders(oID, pID,quatity)
Products(pID, category)
```

We are asked to produce sales statistics for each category of product and return the total quatities that are sold. The schema result should be (`category`, `quantity`).

We are given some information that the buffer size server we have are 500 pages and the size of products is 40 000 pages and the size of orders is unknown but larger than products. We want to join the tables using either SMJ or GHJ.

1.1 What is the range of the size of Orders in which SMJ is equally efficient as GHJ in terms of I/O cost?

The two methods cost is the same when looking at two pass join which is $3(|R| + |S|)$, but they have different requirements for the memory, buffer size. The requirement for the SMJ to be able to complete the algorithm in two phase pass is that the Buffer size $B > \sqrt{|product| + |orders|}$ which is $500 = \sqrt{|40.000| + |orders|}$ we can now see that orders have to be 210.000 or less for the algorithm to be able to run in two passes.

The memory requirement for the GHJ is different however here it is the smaller relation of the two tables that set the limits to the memory requirement so $B > \sqrt{40.000}$. So the SMJ is only as efficient as GHJ if the orders does not exceed 210.000.

1.2 Considering the same sizes as given above, are there sizes of Orders for which Block Nested-Loop Join (BNLJ) has lower I/O cost than SMJ? Briefly illustrate your answer.

The BNLJ makes better use of extra memory compared to the SMJ, but in this case, we are told that orders are larger than products, and when we try to calculate the cost of the two algorithms with a low orders number for the two methods the cost of the BNLJ can be found by the equation

$$|Products| + \frac{|products|}{buffer} \cdot |operations|$$

$$40.000 + \frac{40.000}{500} \cdot 41.000 = 3.320.000$$

while the cost for SMJ can be found by

$$3(|products| + |operation|)$$

$$3(40.000 + 41.000) = 243.000$$

We see here that for this case of chosen numbers that the SMJ is much better than the BNLJ in terms of cost. If the tables however was much smaller but still had the same size buffer then the BNLJ would be better in some cases.

1.3 Describe the most efficient parallel algorithm (in terms of network I/O) that can make use of all the 1 000 machines to carry out this step. What is the cost in terms of network I/O? Briefly explain your answer.

The next step is grouping the records resulting from the previous step by their category values and calculating the sum of quantity values over each group. The final results should be sorted by category. Suppose the previous step produces join results of 200 000 pages, and you have 1 000 machines each with a buffer of 500 pages. Assume the join results have already been randomly partitioned and stored in the 1000 machines.

So with the information given above we have to find the most efficient parrallel algorithm in terms of network I/O that makes use of the 1000 machines to carry out the step. The range partitioning would make sense here and then each partition should be sorted and the perform the aggregation locally. This sort should be done on the category.

2 Question 2: Data Processing II

2.1 State an external memory algorithm with the minimum I/O cost to answer the above query. Argue for the algorithm's correctness and efficiency.

We will use a version of the Two-Phased Multiway Merge-Sort for this problem. This is a common algorithm used for external memory sorting. The algorithm consists of two phases:

1. We sort the data in pieces that has the size of the main memory. We thus create some amount of sublists.
2. We merge the sublists created in the first phase into one single list.

In our case we assume that we can fit B pages into the main memory. The main memory is of size M, that is larger than the square root of the size of player data by at least one page.

Phase 1

In phase 1 we fill up as much of the main memory available to us with a subset of the countries. We sort this subset according to the points of the player. We can use Quicksort to sort this sublist. When the list is sorted, we write it back to the external storage / secondary memory. This is repeated as many times as needed, and we create a sublist after each run.

Phase 2

In phase 2 we will merge the sublists. We will do this by maintaining a main-memory buffer for each sorted sublist and an output buffer for the final sorted list.

As we are provided with a server with memory larger than the square root of the size of the player data by at least one page, we are able to implement an efficient approach in the second phase of this method. We are able to read the first block of each sorted sublist into the main-memory buffer, and this way of handling it proves effective when dealing with a moderately amount of sorted sublists. This allows for a optimized merging process and reducing the number of necessary disk I/O operations.

Now to the merging: Initially, the output buffer is empty. While there are remaining player records in any of the sorted sublists, we do the following:

1. Find the player with the smallest amount of points among the first remaining elements of all of the sublists sorted in phase 1.
2. We move this player to the first available position of the output buffer

(which is empty initially).

3. Now if it happens that the output buffer gets full, it is written to the external storage and is then reinitialized.
4. If the block from which the player with the smallest amount of points was taken is exhausted of records, read the next block from the same sorted sublist into the same buffer.
5. If no blocks remain for a sublist, leave its buffer empty and do not consider players from that list in further competitions for the smallest remaining elements.

We output the grouped information for each country, including player id, country, ranking, and average points, found in the output buffer.

The merging process in phase 1 ensures that records are merged in sorted order. During phase 2, the algorithm identifies the player with the smallest points among the remaining elements in the sorted sublists in each iteration. By doing this in main memory, it efficiently determines which player should be next in the overall sorted list. The integration of grouping and aggregating steps within the merge operation minimizes redundant passes through the data, optimizing the overall efficiency of the algorithm for external memory.

2.2 State the I/O cost of the algorithm you designed. You can make your own notation for the size of each attribute in any table.

The I/O cost is determined by the number of input and output operations during the sorting and merging phases. We define the following: P is the number of pages in the heap file, B is the number of pages that fit in the main-memory.

In the first phase we calculate the cost of the Reads. We need to read the heap file twice, one time when dividing the file into main-memory-sized pieces, and second time when the sorting of the runs are done. Thus this cost us $\frac{2P}{B}$. We then need to write the sorted runs to the external memory, which costs $\frac{P}{B}$.

In the second phase we first do an initial read of the first elements of the sorted runs, then we read the remaining elements of the chosen sorted run. We do this continuously, thus the cost of this is $\frac{2P}{B}$. The cost of writing is the writes to the external memory when the buffer is full. This costs $\frac{P}{B}$. The overall cost is thus:

$$\frac{2P}{B} + \frac{P}{B} + \frac{2P}{B} + \frac{P}{B} = \frac{6P}{B}$$

3 Question 3: Reliability

In this question we assume no correlation between link failures, meaning the failure of one link will not effect the status of any other link in the network. Thus

$$\text{link } i \text{ breaking} \perp\!\!\!\perp \text{link } j \text{ breaking } \forall i \neq j$$

We furthermore assume that the only possible failure in these systems is failure of the links, i.e. machines are not capable of failure, nor electricity or smaller components within these. While unrealistic in practice this is necessary in order to compute probabilities with the given info without having to provide additional information such as probabilities of machine failure, etc.

3.1 What is the probability that the daisy chain network is connecting all the buildings?

Since there are 3 computers connected via two links that each have the probability p of failure, and using the assumed independence of link failures, we find that

$$\begin{aligned} \mathrm{P}\left[\textbf{Daisy chain network failure}\right] &= \mathrm{P}\left[\text{At least one link fails}\right] && (1)\\ &= 1 - \mathrm{P}\left[\text{No links fail}\right] && (2)\\ &= 1 - \left(1 - \mathrm{P}\left[\text{A link fails}\right]\right)^2 && (3)\\ &= 1 - (1-p)^2 && (4) \end{aligned}$$

3.2 What is the probability that the fully connected network is connecting all the buildings?

Since the network is fully connected and has We now have the case where in order for some computer to not be connected to the others, two links need to break instead of only one. Furthermore there are three links in total, so there are $\binom{3}{2} = 3$ different ways of breaking two links. This can be solved by viewing failing of links as a binomial distribution. We thus find that

$$\begin{aligned} \mathrm{P}\left[\textbf{FC network failure}\right] &= \mathrm{P}\left[\text{At least two links fail}\right] && (5)\\ &= 1 - \mathrm{P}\left[\text{No links break}\right] - \mathrm{P}\left[\text{One link breaks}\right] && (6)\\ &= 1 - (1-p)^3 - \binom{3}{2} p(1-p)^2 && (7)\\ &= 1 - (1-p)^3 - 3p(1-p)^2 && (8) \end{aligned}$$

3.3 Which should they purchase?

We are given that for the links in the daisy chain network $\mathrm{P}\left[\text{Daisy link fails}\right] = 10^{-6}$ and the links in the fully connected network $\mathrm{P}\left[\text{FC link fails}\right] = 10^{-4}$ and are asked which of the networks the council should purchase. We find that probability of failure in the daisy chain network is

$$\begin{aligned} \mathrm{P}\left[\textbf{Daisy chain network failure}\right] &= 1 - (1 - p_{\text{daisy}})^2 && (9) \\ &= 1 - (1 - 10^{-6})^2 && (10) \\ &= 2 \cdot 10^{-6} && (11) \end{aligned}$$

We furthermore find the probability of failure in the fully connected network to be

$$\begin{aligned} \mathrm{P}\left[\textbf{FC network failure}\right] &= 1 - (1 - p_{\text{fc}})^3 - 3p_{\text{fc}}(1 - p_{\text{fc}})^2 && (12) \\ &= 1 - (1 - 10^{-4})^3 - 3 \cdot 10^{-4} \cdot (1 - 10^{-4})^2 && (13) \\ &= 2.9997 \cdot 10^{-8} && (14) \end{aligned}$$

And thus we find that it is better to purchase the fully connected network rather than a daisy chained one.