Morales Gonzalez Mikael 234747

CS-422 Database Systems
Project II

Cube Operator (Task 1):

The main difference between the naïve implementation of the cube operator and the optimized version is the stage where the combinations are computed.
The optimized version is done doing a two pass Map-Reduce, where the second pass consist of all possible key combinations with partial results from the previous pass, while the naïve implementation does only one Map-Reduce pass and compute the combination of the keys on each node directly at the beginning, hence it produces a bigger set.
Thus, when the dimension D of the CUBE is increased, the number of combinations increase exponentially, hence so does the running time of both algorithms. The same behavior happens with the input size.
We can vary the number of reducers to distribute the workload across several nodes. In a distributed environment, this will reduce the running time of both algorithms. But, with the naïve implementation, since all the workload is distributed at the beginning, the time spent at each node will be bigger than the optimized version.
The optimized version will distribute less keys at each pass to each node, but will require two passes to compute the result.

This behavior can be seen in my benchmarks, where the running time increase with respect to the input size and the dimension of the CUBE. Nevertheless, since these benchmarks were computed locally on my machine, and not in a distributed environment, we can see that increasing the number of reducers does not reduce the running time of both algorithms of much.
The optimized version would benefit more from a distributed environment, and would logically outperformed the naïve implementation, specially on big datasets.

Theta Join (Task 2):

We can see in the benchmarks for the theta join that, increasing the number of reducers reduce the time needed to join two datasets. But this behavior reaches a plateau as we can see with 40 and 50 reducers. The difference between them is not that big, even though it is faster with 50 reducers.
Similarly, most of the time increasing the max input value decreases the run time execution of the theta join, but again as we can see the difference is minimal between 2000 and 3000 for the max input value.

Increasing the number of reducers distribute the workload across more nodes which internally allows each node to spend less time joining since it is receiving less data.
The max input value allows for bigger buckets. Having a small value for the bucket size would generate too many small buckets, hence it would increase the running time of the algorithm. Creating buckets that are too big would put all the possible candidates in the same bucket which would slow down the algorithm as well.
In my benchmark, the max input is still reasonable, given the number of outputs produce by the theta join, thus as we can see the running time of the algorithm decrease as the max input value increase.

Benchmarks

Cube Operator (Task 1):

- **lineorder_small.tbl**

1 reducer:
Naive Cube 1 dimension: 0.725046721 s
Optimized Cube 1 dimension: 0.304091656 s
Naive Cube with 3 dimensions: 0.793887214 s
Optimized Cube 3 dimensions: 0.533708595 s
Naive Cube with 4 dimensions: 0.839251223 s
Optimized Cube with 4 dimensions: 0.617678549 s

5 reducers:
Naive Cube with 1 dimensions: 0.102684385 s
Optimized Cube with 1 dimensions: 0.17277728 s
Naive Cube with 3 dimensions: 0.303513385 s
Optimized Cube with 3 dimensions: 0.436879231 s
Naive Cube with 4 dimensions: 0.568141061 s
Optimized Cube with 4 dimensions: 0.582110923 s

10 reducers:
Naive Cube with 1 dimensions: 0.084583421 s
Optimized Cube with 1 dimensions: 0.079471385 s
Naive Cube with 3 dimensions: 0.451551469 s
Optimized Cube with 3 dimensions: 0.341109937 s
Naive Cube with 4 dimensions: 0.542141368 s
Optimized Cube with 4 dimensions: 0.69497213 s

- **lineorder_medium.tbl**

1 reducer:
Naive Cube 1 dimensions: 3.172012142 s
Optimized Cube 1 dimensions: 1.981101735 s
Naive Cube with 3 dimensions: 14.002312467 s
Optimized Cube with 3 dimensions: 13.049132722 s
Naive Cube with 4 dimensions: 56.073649661 s
Optimized Cube with 4 dimensions: 52.992031969 s

5 reducers:
Naive Cube with 1 dimensions: 3.018103771 s
Optimized Cube with 1 dimensions: 1.867895464 s
Naive Cube with 3 dimensions: 12.898516741 s
Optimized Cube with 3 dimensions: 15.630634638 s
Naive Cube with 4 dimensions: 49.231117345 s
Optimized Cube with 4 dimensions: 53.061415224 s

10 reducers:
Naive Cube with 1 dimensions: 3.277349467 s
Optimized Cube with 1 dimensions: 1.998656068 s
Naive Cube with 3 dimensions: 13.152940786 s
Optimized Cube with 3 dimensions: 14.861353156 s
Naive Cube with 4 dimensions: 52.28595106 s
Optimized Cube with 4 dimensions: 59.041611546 s

Theta Join (Task 2):

- **5 reducers:**
  - MaxInput of 500
    - 1K input: 7.01892259 s
    - 2K input: 37.352908023 s
  - MaxInput of 1000
    - 1K input: 3.399539186 s
    - 2K input: 27.246501661 s
  - MaxInput of 2000
    - 1K input: 2.350722794 s
    - 2K input: 14.088274211 s
  - MaxInput of 3000
    - 1K input: 1.982168384 s
    - 2K input: 18.921791501 s

- **10 reducers:**
  - MaxInput of 500
    - 1K input: 3.511967903 s
    - 2K input: 25.720966593 s
  - MaxInput of 1000
    - 1K input: 2.15632605 s
    - 2K input: 13.837040829 s
  - MaxInput of 2000
    - 1K input: 1.882306541 s
    - 2K input: 10.832140077 s
  - MaxInput of 3000
    - 1K input: 1.281453757 s
    - 2K input: 13.992576412 s

- **20 reducers:**
  - MaxInput of 500
    - 1K input: 2.457444772 s
    - 2K input: 11.606547685 s
  - MaxInput of 1000
    - 1K input: 2.063062713 s
    - 2K input: 12.078394931 s
  - MaxInput of 2000
    - 1K input: 1.117600431 s
    - 2K input: 9.441946566 s
  - MaxInput of 3000
    - 1K input: 1.691133941 s
    - 2K input: 11.121152216 s

- **50 reducers:**
  - MaxInput of 500
    - 1K input: 2.061230302 s
    - 2K input: 8.869063314 s
  - MaxInput of 1000
    - 1K input: 2.264060705 s
    - 2K input: 7.816102375 s
  - MaxInput of 2000
    - 1K input: 1.492485228 s
    - 2K input: 7.692558925 s
  - MaxInput of 3000
    - 1K input: 1.622980954 s
    - 2K input: 7.548299157 s