Exercise Sheet 2, 2022

6.0 VU Advanced Database Systems

Nicolas Bschor (12132344)

8. April 2022

Aufgabe 1 (Costs of MapReduce) *[1 Punkte]* For this exercise we have to state a view consideration for the base scenario:

We define $|R|$ as the number of input and size($|R|$) as the I/O of reading the input.

- Communication Cost c: c = size($|R|$) (Output very small -> not significantly effecting the Communication Cost)
- Replication Rate rr: $1 \gg rr \geq \frac{k}{|R|}$ (Replication Rate roughly around the ideal replication rate of $\frac{k}{|R|}$ and much smaller than 1. Every Node emits a bit more than it's top k values, dependent on the data)
- Reducer Size rs: $1 \gg rs \geq k \cdot$ nodes (Every node emits a bit more than it's top k tuples to the key "None"'is mapped on $\geq k \cdot$ nodes elements)

(a)
- Communication Cost c: c = size($|R|$) (Only emit more tuples, so there is not more I/O) $\rightarrow$ Same as base scenario
- Replication Rate rr: $rr = 1$ (For every input emit one tuple) $\rightarrow$ Much higher than base scenario
- Reducer Size rs: $|R|$ (Every Node emits it's input on "None"'key)

(b) Every node emits $\approx k$ values on a specific pre-reducer. After the pre-reducing the result is then mapped to the "None" key for the final reducing. We consider a useful distribution of the pre-reducers (e.g nodes 1-10 to reducer 1, 11-20 to reducer 2...).

- Communication Cost c: c = size($|R|$) + [prereducers $\cdot * k + k$] (prereducers $\cdot * k$: I/O to write the result of pre-reducers (not high); k: write final result) $\rightarrow$ A bit higher than base scenario.
- Replication Rate rr: For both jobs, the same replication rate as in the base scenario.
- Reducer Size rs: $1 \gg rs \geq k \cdot \frac{\text{nodes}}{\text{pre-reducers}}$, $1 \gg rs \geq k \cdot$ pre-reducers, $\rightarrow$ smaller than base scenario.

- Skew: Since we split the emitted keys evenly on the pre-reducers we have as much keys as pre-reducers and all should contain ≈ the same size of the values → No skew

(c)
- Communication Cost c: $c = \text{size}(|R|) + \text{size}(k)$ (Now the number of k is high, so the I/O for writing can be a factor as well)
- Replication Rate rr: $rr = \frac{k}{|R|}$ (We emit a lot more tuples since we search for more values) → Much higher than base scenario
- Reducer Size rs: nodes $* k \leq rr$ (since k is very high we have a lot of values mapped to "None") → higher than base scenario

Aufgabe 2 (Relational Operations) *[2 Punkte]*

(a)

$$F_1(R, S) := \sigma_{A+B=C}(R) \bowtie (\sigma_{B>3}(S) \ltimes \sigma_{C>B}(R)) = \sigma_{A+B=C \land C>B}(R) \bowtie \sigma_{B>3}(S)$$

Solution with a single map/ reducer pair.

Mapper:

When tuple t of S: If $t.B > 3$ then emit the key/value pair:
<($t_S.B$, $t_S.C$), ("S", t_S)>

When tuple t of R: If $t.C > t.B \land t.A + t.B = t.C$ then emit the key-value pair:
<($t_R.B$, $t_R.C$), ("R", t_R)>

Reducer:

Build two buffers of the values: Buffer one with the ("R", t_R) values and buffer 2 with the ("S", t_S) values. Then make a nested loop with the two buffers and emit the "cross product" of the two buffers. Key-value pair: (_, ($t_R \circ t_S$))

(b) 1) Job:

Mapper:

If tuple $t \in T$: emit <($t.B, t.C$), $t.A$>
If tuple $t \in S$: emit <NULL, ($t.B, t.C$)>

Reducer:

If key = null: Sort values and emit <($t_1.B, t_1.C, t_2.B, t_2.C, ..., t_n.B, t_n.C$), True> with $t_1, t_2, ..., t_n$ as the values.
If key ∈ A: Sort values and emit all lengths of the keys with the following form:
<(($t_1.B, t_1.C$)), key>
<(($t_1.B, t_1.C$), ($t_2.B, t_2.C$)), key>
...

<$((t_1.B, t_1.C), (t_2.B, t_2.C), ..., (t_n.B, t_n.C))$, key>

2) Job:

Reducer:

If True in the Values: emit all other values

Example: U := [2,2], [1,1], [1,2]
T := [A1, 1,1], [A1, 1,2] [A1, 2,2], [A1, 3,3], [A2, 1,1], [A3, 3,3]
1. Job Mapper:
From U:
Null-> [[2,2], [1,1], [1,2]]
From T: A1 -> [[2,2], [1,1], [1,2]]; A2 -> [1,1]; A3 -> [3,3]

1. Job Reducer:
From Null: ([1,1], [1,2], [2,2]) -> True
From A1: ([1,1]) -> A1; ([1,1], [1,2]) -> True; ([1,1], [1,2], [2,2]) -> True;
([1,1], [1,2], [2,2], [3,3]) -> True
From A2: [1,1] -> A2
From A3: [3,3] -> A3

2.Job Reducer: ([1,1], [1,2], [2,2]) -> True, A1 => emit A1

Aufgabe 3 (MapReduce in Practice) *[4 Punkte]* Silence is gold

Aufgabe 4 (Hive) *[4 Punkte]* Silence is gold

Aufgabe 5 (Spark in Scala) *[4 Punkte]*

(a) File: "Exercise5_SparkInScala.ipynb"

(b) First the csv file of **"customers"** is scanned. Then a filter process is executed and the filtered tuples are then exchanged and sorted afterwards. All these dependencies are **narrow dependencies**.

Now the csv file of **"articles"** is scanned. Then it is filtered (**narrow transformation**) and broadcasted afterwards (**narrow dependency**).

Then the csv file of **"transactions"** is scanned. It's also filtered through a (**narrow transformation**) and afterwards BrodcastHashJoined with the results from "articles" (**wide dependency**).

The joined result of "articles' and "transactions" it then exchanged and sorted (**narrow dependency**).

Now a **narrow dependent** SortMergeJoin can be applied the joined tuples of "articles" & "transactions" and "customers".

With the join result two **narrow dependent** HachAggregate can be executed, to emit the final result.