2AMD15 - Big Data Management

Calli Evers Sander Cauberg Filippo Daniotti Klará Tauchmanová
Horea Breazu Filipe da Costa Quinta Goncalves Sobrinho
Eindhoven University of Technology

Question 2

SQL query for Question 2:

```
SELECT CONCAT_WS('-', t1._1, t2._2, t3._3) AS full_id,
       var_udf(ARRAY(t1._1, t2._2, t3._3)) AS var
FROM (
    SELECT * FROM (
        SELECT * FROM (
            SELECT _1 FROM dataframe) t1
            CROSS JOIN (
                SELECT _1 AS _2 FROM dataframe) t2
        WHERE t1._1 < t2._2) t12
        CROSS JOIN (
            SELECT _1 AS _3 FROM dataframe) t3
    WHERE t12._2 < t3._3)
WHERE var < τ
```

Optimization techniques:

- Fine-tuning the usage of partitions
- Broadcast <id, vector> map to reduce network consumption
- Usage of `numpy.ndarrays` with 16 bits integers

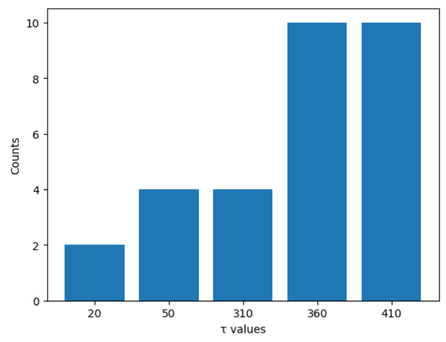

Figure 1. Triples with aggregate variance $\leq \tau$

Optimization	Execution time
All	163.56s
No numpy	443.19s
No repartition/coalesce	2108.38s
No broadcast	ran out of heap

Table 1. Impact of optimizations tricks on the dataset of size 125x10000, ran locally

Partitions	Execution time
32-64-128	**237.11s**
16-32-64	246.13s
64-128-256	295.03s
64-64-64	323.13s

Table 2. Impact of partition combinations on the dataset of size 250x10000, ran **on the server**

Question 3

System architecture for Question 3:

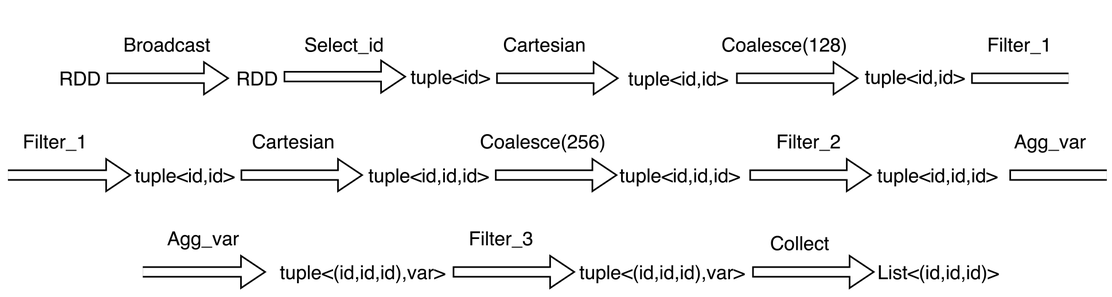

Figure 2. Brief description of system architecture for Q3

Optimization techniques:

- Fine-tuning the usage of partitions
- Broadcast <id, vector> map to reduce network consumption
- Usage of `numpy.ndarrays` with 16 bits integers
- Usage of `.Accumulator()` API
- Caching filtered triples with $var \leq 410$ before filtering again for $var \leq 20$ *(discarded)*
- Pre-computing the mean of each vector to speed up `statistics.pvariance()` *(discarded)*

Optimization	Execution time
All	30.59s
No numpy	2940.19s
No broadcast	185.30s
No accumulator	36.44s
No coalesce	131.46s

Table 3. Impact of optimizations tricks on the dataset of size 250x10000, ran locally

Optimization	Execution time
64-128-256	**910.05s**
128-256-512	946.79s
125-250-500	969.51s
64-64-128	1005.65s
25-50-100	1097.70s
64-64-64	1122.74s
128-128-128	1162.01s

Table 4. Impact of partition combinations on the dataset of size 1000x10000, ran **on the server**

Question 2 vs. Question 3

Both solution produced the **same results**:

	Q2 code	Q3 code
Results $\tau = 20$	2	2
Results $\tau = 410$	10	10
Execution time	**237.11s**	**42.35s**

Table 5. Comparison between Q2 code and Q3 code on the same dataset (250 vectors)

References

Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):5875, apr 2005. ISSN 0196-6774. doi: 10.1016/j.jalgor.2003.12.001. URL https://doi.org/10.1016/j.jalgor.2003.12.001.

Question 4

We used **Count-min (CM) sketch** (Cormode and Muthukrishnan [2005]). Illustration of the approximation of the aggregate variance:

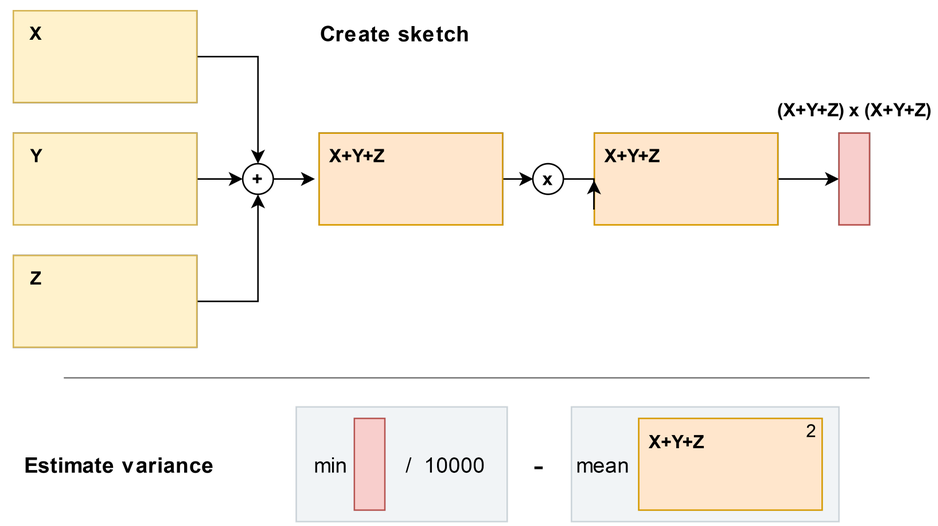

Figure 3. Count-min sketch exploitation for Q4

Theorem 1. $\sigma^2 \leq \widehat{\sigma^2}$ and, with probability $1-\delta$, we have that $\widehat{\sigma^2} \leq \sigma^2 + \frac{1}{10000}\varepsilon||\mathbf{a}||_1^2$

Proof. By the definition of $\widehat{\sigma^2}$ and σ^2 and from the **Theorem 3** from the Cormode and Muthukrishnan [2005] paper

$$\sigma^2 = \frac{1}{10000}(\mathbf{a} \odot \mathbf{a}) - \mu_{\mathbf{a}}^2 \leq \frac{1}{10000}\widehat{(\mathbf{a} \odot \mathbf{a})} - \mu_{\mathbf{a}}^2 = \widehat{\sigma^2} \tag{1}$$

Therefore, $\sigma^2 \leq \widehat{\sigma^2}$ for non-negative vectors. Similarly,

$$\begin{aligned}\mathbb{P}\left[\widehat{\sigma^2} - \sigma^2 \geq \frac{1}{10000}\varepsilon||\mathbf{a}||_1^2\right] &= \mathbb{P}\left[\frac{1}{10000}\left(\widehat{(\mathbf{a} \odot \mathbf{a})} - (\mathbf{a} \odot \mathbf{a})\right) \geq \frac{\varepsilon||\mathbf{a}||_1^2}{10000}\right] \\ &= \mathbb{P}\left[\left(\widehat{(\mathbf{a} \odot \mathbf{a})} - (\mathbf{a} \odot \mathbf{a})\right) \geq \varepsilon||\mathbf{a}||_1^2\right] \leq \delta\end{aligned} \tag{2}$$

So, $\mathbb{P}\left[\widehat{\sigma^2} - \sigma^2 \geq \frac{1}{10000}\varepsilon||\mathbf{a}||_1^2\right] \leq \delta$, as required.

QED

System architecture for Question 4:

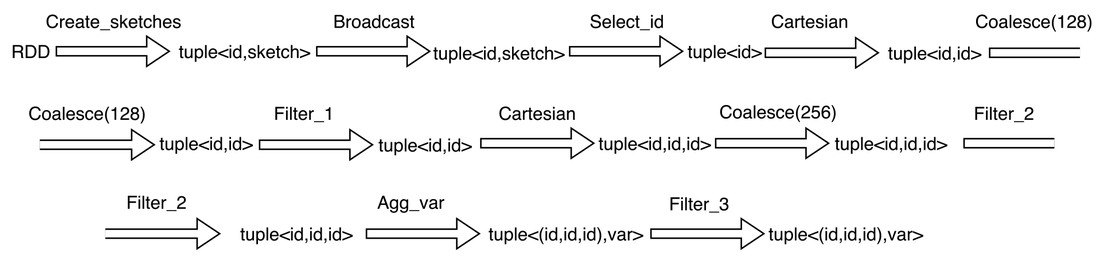

Figure 4. Brief description of the system architecture for Q4

The execution time of Q4 on the cluster: **164.33 seconds**

	$\varepsilon = 0.0001$	$\varepsilon = 0.001$	$\varepsilon = 0.002$	$\varepsilon = 0.01$
$\tau \leq 400$	-	$P = 1^1_;R = 0/10$	-	$P = 1^1_;R = 0/10$
$\tau \geq 200000$	$P = 1, R = 1$	$P = 1/26819, R = 1$	$P = 1/91885, R = 1$	$P = 1/2573000, R = 1$
$\tau \geq 1000000$	$P = 1^1_;R = 1^2$	$P = 1^1_;R = 1^2$	$P = 0/1, R = 1^2$	$P = 0/91885, R = 1^2$

[1] Number of TPs is 0, number of FTs is also 0 (eg. 0/0)
[2] Number of relevant triplets is 0, number of retrieved items is also 0 (eg. 0/0)

Table 6. Precision/recall values of the Q4 results.

CM sketch always **overestimating** the results, therefore:

- for *functionality 1* ($\leq \tau$): counts are underestimated, only TP triplets $\rightarrow$ precision is 1
- for *functionality 2* ($\geq \tau$): counts are overestimated, lots of FP triplets, but no FNs $\rightarrow$ recall is 1
- smaller $\delta \rightarrow$ larger depth $d \rightarrow$ a better approximation of the aggregate variance $\rightarrow$ higher recall for *functionality 1* and higher precision for *functionality 2*
- for smaller $\varepsilon \rightarrow$ larger width $w \rightarrow$ fewer collisions $\rightarrow$ a better approximation $\rightarrow$ higher recall for *functionality 1* and higher precision for *functionality 2*

The dimension of the sketch is important:

- too small $\varepsilon \rightarrow$ huge size of the sketch $\rightarrow$ slow computation, but more precise results
- too big $\varepsilon \rightarrow$ small size of the sketch $\rightarrow$ fast computation, but high number of FPs

The tightness of the bounds:

- *lower bound*: tight
- *upper bound*: vary for each aggregate vector, the error is relative to the first norm squared

Ethical and other aspects

- *Relational databases* should be employed if the size of the problem is small, because:
  - they are more widely known (thus maintainable)
  - no need to add complexity if the problem is small
- *Multi-threaded programs* should be avoided, because:
  - it does not provide developers with any advantage over Spark
  - Spark provides users with better scalability
- *Company-owned cluster vs. cloud resources* - the choice is a trade-off between:
  - hardware/resources availability of the company
  - sensitivity of the data/context
- *Spark vs. approximation technique* - the choice depends on:
  - hardware/resources availability of the company
  - precision/recall requirements of the company for the problem