Deep Diving in the Data Lake using Vector-H

A Seminar Report

Submitted by

KALINGA BHUSAN RAY

Technische Universität Ilmenau

Databases and Information Systems Group

Abstract:

VectorH is a world class product from Actian, VectorH built natively to run on Hadoop clusters and supports analytic workload. As part of this Seminar task, 3 of the Amplab queries were run on 15-node hadoop cluster with VectorH installed. Query execution time is measured and compared for various constellations and strategies.

DataSet:

Two Datasets from Amplabs were used for this task.

1. Rankings: Lists websites and their page rank, with the below schema
   pageURL VARCHAR(300), pageRank INT, avgDuration INT

2. UserVisits: Stores server logs for each web page with the below schema
   sourceIP VARCHAR(116), destURL VARCHAR(100), visitDate DATE
   adRevenue FLOAT, userAgent VARCHAR(256), countryCode CHAR(3)
   languageCode CHAR(6),searchWord VARCHAR(32),duration INT

Queries: Out of 4 Amplab queries first 3 queries were executed on the provided data set.

Query1: SELECT pageURL, pageRank FROM rankings WHERE pageRank > X

Query2: SELECT SUBSTR(sourceIP, 1, X), SUM(adRevenue) FROM uservisits GROUP BY
SUBSTR(sourceIP, 1, X)

Query3:

```
SELECT sourceIP, totalRevenue, avgPageRank
FROM
  (SELECT sourceIP,
        AVG(pageRank) as avgPageRank,
        SUM(adRevenue) as totalRevenue
    FROM Rankings AS R, UserVisits AS UV
    WHERE R.pageURL = UV.destURL
      AND UV.visitDate BETWEEN Date(`1980-01-01') AND Date(`X')
```

GROUP BY UV.sourceIP)
ORDER BY totalRevenue DESC LIMIT 1

Query1 deals with scanning of the table data, Query2 deals with aggregation and Query3 is a Join query. Value for the X for each query initially chosen and got fixed for further runs with different constellation enabling compare and contrast the results. For query 1 X is fixed as 100, for query 2 10 and for query 3, its 1970-01-01.

Data Storage: With help of VectorH, data can be stored in the internal table or an external table can be created having reference to the existing data. In general same query gives different executing time when operated on internal or external table. Also data loading time also another parameter to be considered while comparing the query loads.

Data Loading: Loading data from the HDFS to the internal table using sparkLoader from Actian takes below timings captured using time built-in utility from Linux. Total preparation time for the internal table is 2 m 17.898s (0m41.942s for Rankings and 1m35.956s for UserVisits).

Rankings	Uservisits
real 0m41.942s user 0m18.072s sys 0m1.104s	real 1m35.956s user 0m18.604s sys 0m1.140s

Query Execution Times:

1. Without any special strategy

Int./Ext. table	Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost
Internal	Cold	0m3.718s	0m15.359s	2m53.870s	3m12.947s
Internal	Hot	0m3.074s	0m12.291s	2m38.362s	2m53.727s
External	Cold	0m6.811s	0m28.238s	2m48.581s	3m23.63s
External	Hot	0m6.669s	0m22.321s	2m48.682s	3m17.672s

2. Creating table with 15 partitions

Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost
Cold	0m3.580s	0m12.328s	2m47.791s	3m3.699s
Hot	0m3.505s	0m12.268s	2m35.849s	2m51.622s

3. Creating table with 15 partitions, considering Join attribute(HASH ON pageURL and destURL)

Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost
Cold	0m3.598s	0m12.042s	2m36.185s	2m51.825s
Hot	0m3.520s	0m11.862s	2m36.294s	2m51.672s

4. Creating Index on strategic table attribute(pageURL, sourceIP)

Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost
Cold	0m3.584s	0m12.938s	2m48.036s	3m4.558s
Hot	0m3.574s	0m12.574s	2m36.174s	2m52.322s

5. Index on ON UserVisits using (visitDate, destURL)

Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost
Cold	0m3.447s	0m12.037s	2m45.222s	3m0.706s
Hot	0m3.417s	0m12.051s	2m37.191s	2m52.659s

6. Creating Statistics

Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost
Cold	0m3.488s	0m11.929s	2m36.214s	2m51.631s
Hot	0m3.407s	0m12.157s	2m35.907s	2m51.134s

7. Invoking Optimize DB

Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost
Cold	0m3.549s	0m12.136s	2m36.494s	2m52.089s
Hot	0m3.436s	0m12.539s	2m36.147s	2m52.122s

8. Invoking Parallelization with 30

Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost
Cold	0m3.415s	0m12.517s	2m37.476s	2m53.408s
Hot	0m3.259s	0m12.152s	2m36.874s	2m52.285s

9. Invoking Parallelization with 60

Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost

Cold	0m3.623s	0m13.742s	2m36.102s	2m53.467s
Hot	0m3.395s	0m12.004s	2m36.080s	2m51.479s

10. Invoking Parallelization with 90

Cold/Hot	Query 1	Query 2	Query 3	Cumm. Cost
Cold	0m3.838s	0m14.171s	2m35.698s	2m53.707s
Hot	0m3.584s	0m12.333s	2m36.339s	2m52.256s

Cost analysis:

- Cumulative cost:

Least cumulative cost for internal table including preparation time is 5m9.529s for cold and 5m9.032s for hot execution. Least cumulative cost for external table for cold and 3m23.63s and 3m17.672s for hot execution.

- Maximum cost:

For internal table, minimum cold execution time for query 1 including preparation time is 45.357s (Option 8), minimum hot execution time for query 1 observed to be 45.201s which is comparatively higher than corresponding external table(cold time 0m6.811s). So hot query execution against external table exhibited better performance with less execution time for all 3 queries.

Conclusion:

We can observe that with Partitioning, execution time got reduced and when applying partitioning on join attributes, execution time also reduced further. Considering creation of indexes on attribute(pageURL, sourceIP) and (visitDate, destURL) was not much helpful, but it is observed that with having an index for the table UserVisit on (visitDate, destURL) performed little better than having index on the attribute sourceIP.

Using other features of the VectorH such as *Creating Statistics* improved execution time mainly for query 3; however *Optimize DB* almost did not add anything to the improvements for the given set of query execution. It is also noticed that when using parallelization performance improves with number of threads from 30 to 90 and when number of threads again increased to 120, the performance decreased. In comparison to external table, query execution with internal table performs significantly better for query 1 and 2, and slightly better for query 3. Hot execution of query which reuses much of the loaded buffers e.g. query 3 exhibits better results in comparison to query 1 and 2. It is observed that there are different techniques available to speed up query execution, sometimes depending on the given dataset and query, different techniques yield different degree of optimization.

References:

[1] Amplab Big Data Benchmark, https://amplab.cs.berkeley.edu/benchmark/

[2] Vector in Hadoop 5.0, http://docs.actian.com/vectorhadoop/5.0/index.html

[3] Spark-Vector connector, https://github.com/ActianCorp/spark-vector