Using Vectorized Execution to Boost SQL Query Performance on Spark

Yijie Shen†‡, Jin Xiong†‡, Dejun Jiang†‡

† Institute of Computing Technology, CAS, Beijing, China
‡ University of Chinese Academy of Sciences, Beijing, China
The pursuit of high-performance query processing

• People never get satisfied with data processing speed
  • General-purpose DBMS
  • OLAP (HyPer / MonetDB)
  • SQL-on-Hadoop (MPP / MapReduce-based SQL engines)

• The speed of the Spark engine itself is also accelerating
  • ~3x speed up since its first release
Physical-Op is the basic execution unit

```
SELECT a_1, SUM(5 * a_2), AVG(b_2 + 4)
FROM A, B
WHERE A.a_1 = B.b_1 AND A. a_2 > 10;
```
The execution mode for Physical-Op matters

Volcano model

Data-centric compilation

In-register Processing

Complex-Op not changed

In-Cache Processing

All operators run in vec-mode

Too much function call

Too much branching

50th International Conference on Parallel Processing (ICPP)
August 9-12, 2021 in Virtual Chicago, IL
Breaking-down the benchmark execution time

Spark SQL TPC-H Exec-time breakdown

- Simple Op 13%
- Shuffle 21%
- Filter/Project
- Sort/Agg/Join
- Complex Op 66%

Amplification of shuffle-data footprint with naïve vectorized shuffle

Sub-optimal batch length lead to CPU cache under-utilized

Too much random memory access
VEE architecture

SQL Query

Driver

Query Parser → Query Optimizer → Physical Planner → Vectorization Rewriter → DAG Scheduler

Executors

Vectorized Projection → Vectorized Shuffle → Vectorized Aggregate → Vectorized Merge-Join → Vectorized Sort

Vectorized Reader → Vectorized Sort → Parquet File

HDFS

Vectorized Shuffle → Vectorized Projection

Vectorized Assembler

Text File

50th International Conference on Parallel Processing (ICPP)
August 9-12, 2021 in Virtual Chicago, IL
Vectorized Data Structures

UnsafeRow in vanilla Spark

null bits | Int | Byte | Float | Long
----------|-----|------|-------|------

8 byte 8 byte 8 byte 8 byte 8 byte

Record batch Runtime Repr

Record Batch Header
- Capacity
- Size
- numCols
- Column Vectors Ptr

Column Vector Array
- Vector_1_Ptr
- Vector_2_Ptr
- Vector_n_Ptr

Column Vector n
- Data Array Ptr
- Null Array Ptr

Data Array

Null Array

Record batch In memory & disk

# Row
- # Column Vector
- # Vector_1_null
- Vector_1_data size
- Vector_1_null_1_pos
- Vector_1_null_2_pos

# Vector_2_null
- Vector_2_data size
- Vector_2_null_1_pos
- Vector_2_null_2_pos

# Vector_n_null
- Vector_n_data

Address Space

(a) (b)
Vectorized Shuffle based on Serialization-aware assembling

• Naïve vectorized shuffle
  • Metadata takes too much space
  • Shuffle data amplification
    • CPU, I/O inefficiency

• Serialization-aware assembling
  • assembling overhead is significantly reduced
  • the advantages of vectorized shuffle is maintained.
Random memory accesses are greatly reduced

50th International Conference on Parallel Processing (ICPP)
August 9-12, 2021 in Virtual Chicago, IL
Operator-aware batch length

Existing method:
Keeping the whole batch in cache

\[ BL_{rb} = \frac{LLC \text{ Size}}{\sum_i rb.\text{Vec}[i].\text{FieldSize}} \]

Observation:
Only part of the vectors are used at each step during the computation of complex-op

Only need to guarantee each step’s vectors cache-residential

Aggressive batch length

\[ BL_{rb} = \frac{LLC \text{ Size}}{4 + \max_j (\sum_i rb.\text{KeyVec}[i].\text{FieldSize}, \text{rb.Vec}[j].\text{FieldSize})} \]

Example: Sort

- Step 1: in-cache sort
  - Row number vec. and
  - Key vecs
- Step 2: partial merge
  - Row number vec. and
  - one vec.
Experiment Setup

• Workload: TPC-H all 22 queries

• Configuration
  • Four-machine Spark cluster
  • Two Intel Xeon E5645 processors, 32GB of memory and eight 1TB HDDs
  • Each processor has 6 cores, 12 hyper threads and 12MB LLC
  • Linux CentOS 7.3 with kernel 3.10.0

• TPC-H 1TB data
Shuffle performance

Improves execution time by up to 1x and 63.9% on average than Spark

VEE greatly reduces the data footprint by up to 51.8% and 29.6% on average compared to Spark due to SAA
Sort performance

Select \( SC_1, \cdots, SC_k, \; OC_1, \cdots, OC_j \)

from range(start = 0, end = num, p = 24)

order by \( SC_1, \cdots, SC_m; \)

Partial merges with rearrangement save 47.8% time on average compared to their counterparts without rearrangement.

VSTM increase much less (by 37%, 96% and 1.9x respectively), and spilling only takes 16% of total time on average, benefiting from compact batch layout.

The in-cache sort time decreases as the batch length grows until the batch length reaches 128 K records.
Overall TPC-H performance

The performance speedup of VEE against Spark is up to 72.7% and 25.0% on average.
Thanks

Using Vectorized Execution to Boost SQL Query Performance on Spark

Yijie Shen