Parquet-MR acceleration using VectorAPI

Jatin Bhateja (Intel)
Fang Xie (Intel)

Bitpack algorithm is very popular in database and big data storage system like Parquet, ORC. Vector API optimization provide better performance for en/decoding bitpack.

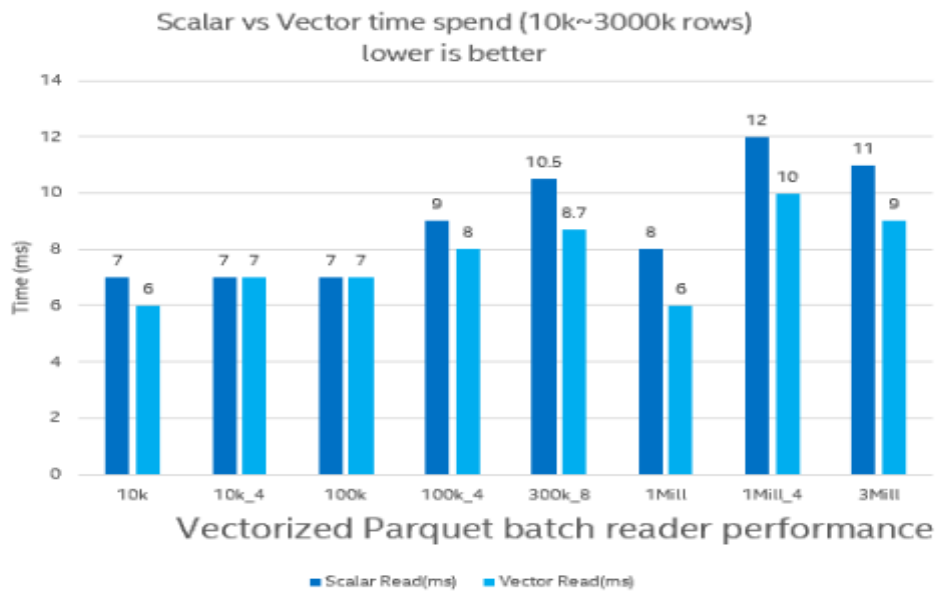
Below are performance impovement reports

Bit Width	1	2	3	4	5	6	7	8	9	10
Scalar time (ms):	3193	163	145	81	99	99	100	38	91	96
Vector time (ms):	873	28	41	15	25	19	17	8	12	12
Performance gain	3.66x	5.82x	3.53x	5.4x	3.96x	5.21x	5.88x	4.75x	7.58x	8.0x

Vectorized Execution: bit-packing decode

Test2： Reader test

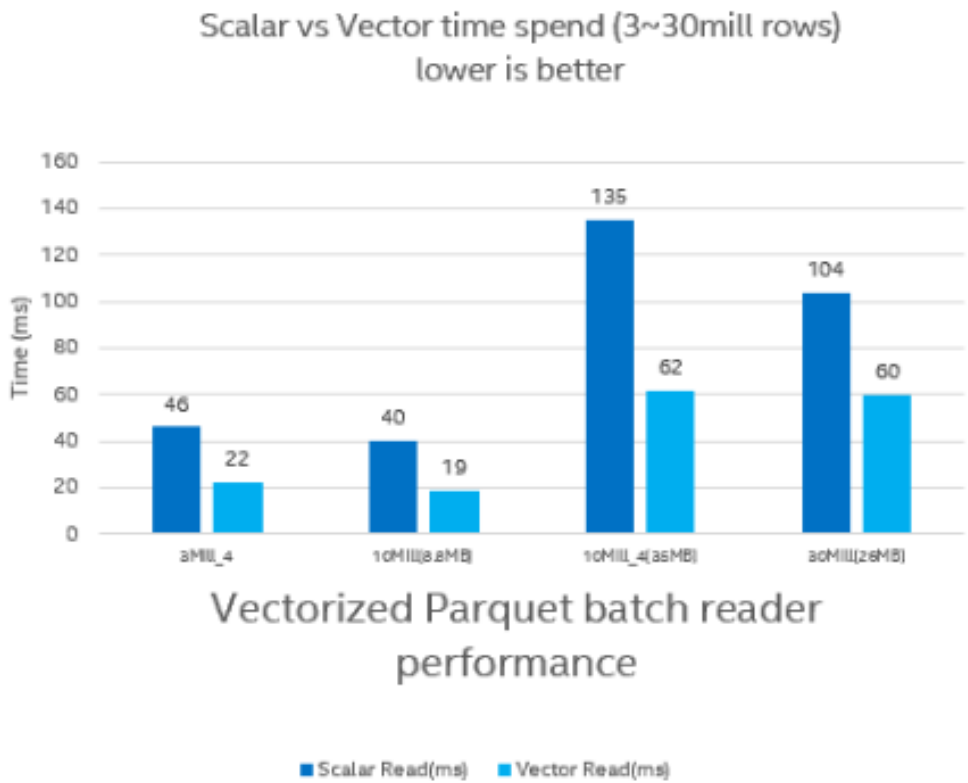
- Location: Spark VectorizedParquetRecordReader
- Read mode: Batch column reader
- Bit width: 7
- Compression: false
- Batch Size: 4096
- Column: 1~4 columns
- Rows: 10k~3000k



Vectorized Execution: bit-packing decode

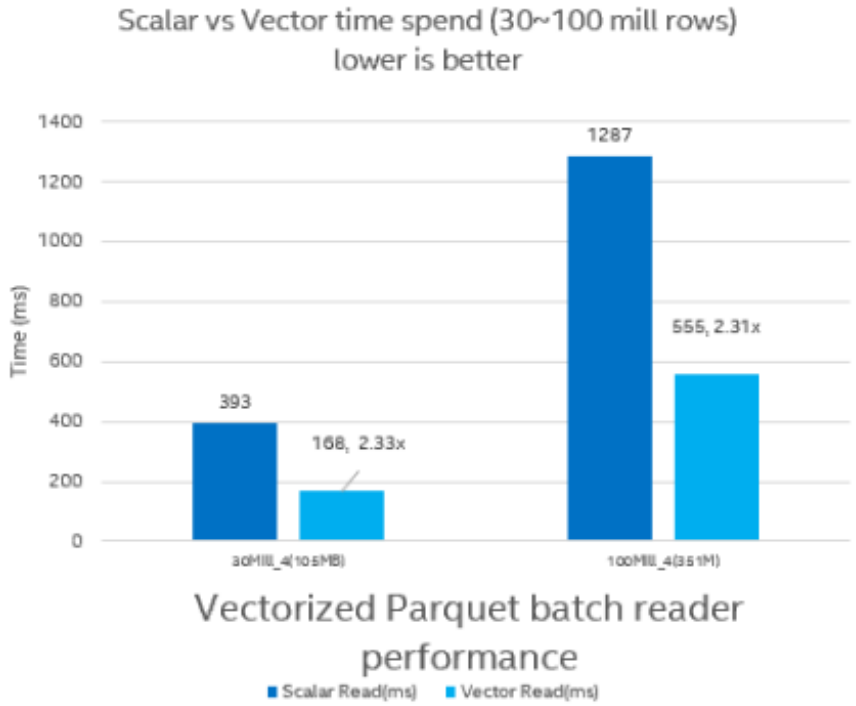
Test2： Reader test

- Location: Spark VectorizedParquetRecordReader
- Read mode: Batch column reader
- Bit width: 7
- Compression: false
- Batch Size: 4096
- Column: 1~4 columns
- Rows: 3Mills~30Mills



Vectorized Execution: bit-packing decode

Test2： Reader test
- Location: Spark VectorizedParquetRecordReader
- Read mode: Batch column reader
- Bit width: 7
- Compression: false
- Batch Size: 4096
- Column: 1~4 columns
- Rows: 30Mills~100Mills



Test sql query in Spark3.3.0 with OpenJDK17， the bitpack optimization improved sql query by 2x~3x in some special case (deep scan file cases)

Spark SQL query with the optimization of parquet

Hardware configuration

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                112
On-line CPU(s) list:   0-111
Thread(s) per core:    2
Core(s) per socket:    28
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
BIOS Vendor ID:        Intel(R) Corporation
CPU family:            6
Model:                 106
Model name:            Intel(R) Xeon(R) Platinum 8372C CPU @ 3.20GHz
BIOS Model name:       Intel(R) Xeon(R) Platinum 8372C CPU @ 3.20GHz
Stepping:              6
CPU MHz:               3500.000
CPU max MHz:           3500.0000
CPU min MHz:           800.0000
BogoMIPS:              6400.00
Virtualization:        VT-x
L1d cache:             48K
L1i cache:             32K
L2 cache:              1280K
L3 cache:              43008K
NUMA node0 CPU(s):     0-27,56-83
NUMA node1 CPU(s):     28-55,84-111
```

Spark configuration

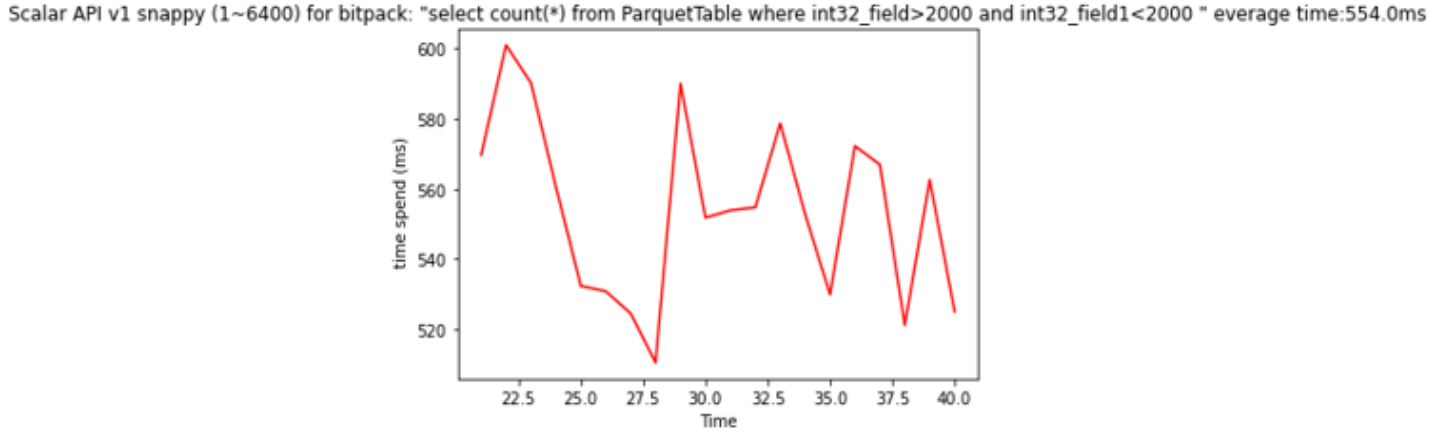
Standalone cluster 1 host
1 master
1 worker
Master memory: 2g
Executor memory:32g
Driver memory:5g

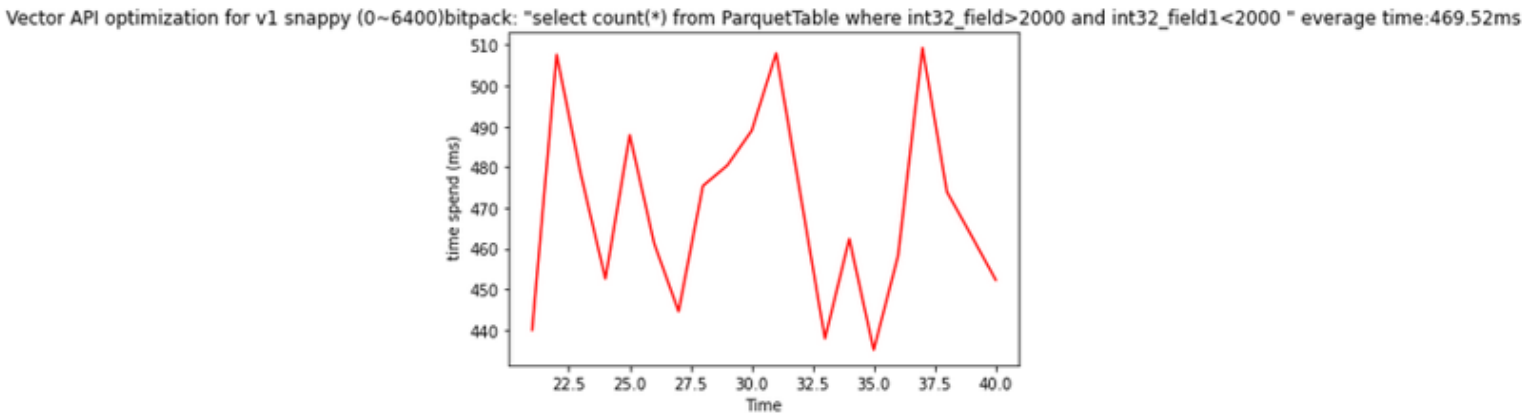
Spark SQL query with the optimization of parquet

select count(*) from ParquetTable where int32_field>2000 and int32_field1<2000

Test1 ： SQL test

- Location: Spark SQL
- Data type: INT32
- Compression: Snappy
- Column:4 columns
- Rows: 100000000
- File size: 652M
- Perf gain: 15%





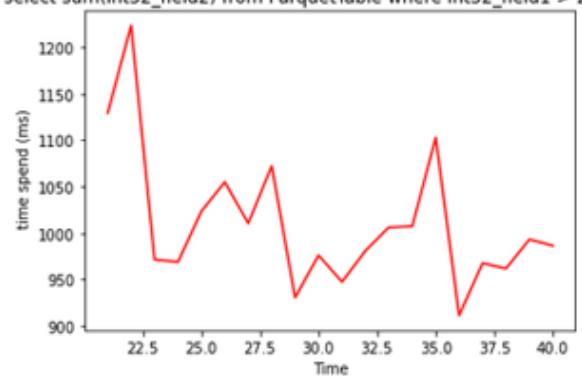
Spark SQL query with the optimization of parquet

select sum(int32_field2) from ParquetTable where int32_field1 > 2000 group by int32_field2

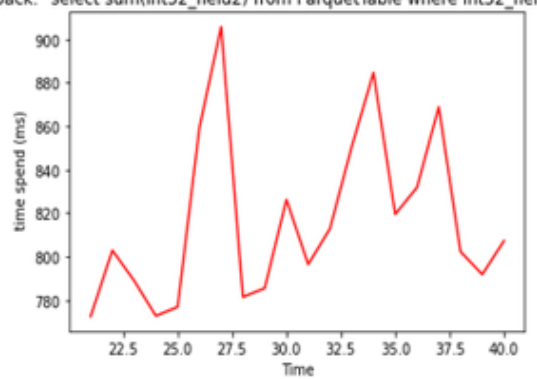
Test1 ： SQL test

- Location: Spark SQL
- Data type: INT32
- Compression: Snappy
- Column:4 columns
- Rows: 100000000
- File size: 652M
- Perf gain: 19%

Scalar API v1 snappy (1~6400) for bitpack: "select sum(int32_field2) from ParquetTable where int32_field1 > 2000 group by int32_field2" average time:1011.31ms



Vector API optimization for v1 snappy (0~6400)bitpack: "select sum(int32_field2) from ParquetTable where int32_field1 > 2000 group by int32_field2" everage time:816.92ms



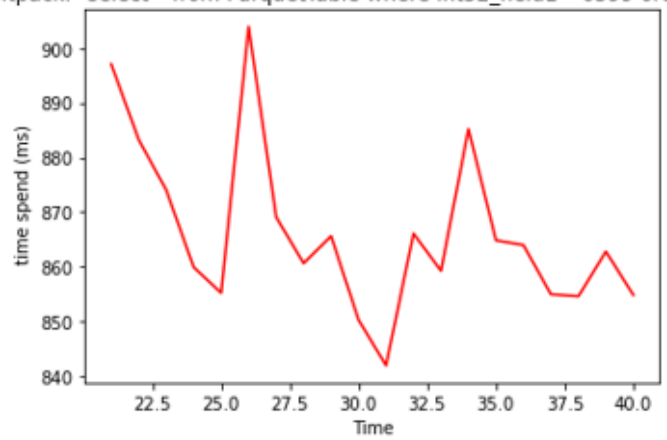
Spark SQL query with the optimization of parquet

select * from ParquetTable where int32_field1 =6399 order by int32_field2

Test1 ： SQL test

- Location: Spark SQL
- Data type: INT32
- Compression: Snappy
- Column:4 columns
- Rows: 100000000
- File size: 652M
- Perf gain: 20%

Scalar API v1 snappy (1~6400) for bitpack: "select * from ParquetTable where int32_field1 =6399 order by int32_field2" average time:866.38ms



Vector API optimization for v1 snappy (0~6400)bitpack: "select * from ParquetTable where int32_field1=6399 order by int32_field2" everage time:675.79ms