Vectorized Query Execution

- Introduction
- Using Vectorized Query Execution
  - Enabling vectorized execution
  - Supported data types and operations
  - Seeing whether vectorization is used for a query
- Limitations
- Version Information

Introduction

Vectorized query execution is a Hive feature that greatly reduces the CPU usage for typical query operations like scans, filters, aggregates, and joins. A standard query execution system processes one row at a time. This involves long code paths and significant metadata interpretation in the inner loop of execution. Vectorized query execution streamlines operations by processing a block of 1024 rows at a time. Within the block, each column is stored as a vector (an array of a primitive data type). Simple operations like arithmetic and comparisons are done by quickly iterating through the vectors in a tight loop, with no or very few function calls or conditional branches inside the loop. These loops compile in a streamlined way that uses relatively few instructions and finishes each instruction in fewer clock cycles, on average, by effectively using the processor pipeline and cache memory. A detailed design document is attached to the vectorized query execution JIRA, at https://issues.apache.org/jira/browse/HIVE-4160.

Using Vectorized Query Execution

Enabling vectorized execution

To use vectorized query execution, you must store your data in ORC format, and set the following variable as shown in Hive SQL (see Configuring Hive):

```sql
set hive.vectorized.execution.enabled = true;
```

Vectorized execution is off by default, so your queries only utilize it if this variable is turned on. To disable vectorized execution and go back to standard execution, do the following:

```sql
set hive.vectorized.execution.enabled = false;
```

Additional configuration variables for vectorized execution are documented in Configuration Properties – Vectorization.

Supported data types and operations

The following data types are currently supported for vectorized execution:

- `tinyint`
- `smallint`
- `int`
- `bigint`
- `boolean`
- `float`
- `double`
- `decimal`
- `date`
- `timestamp` (see Limitations below)
- `string`

Using other data types will cause your query to execute using standard, row-at-a-time execution.

The following expressions can be vectorized when used on supported types:

- arithmetic: `+`, `-`, `*`, `/`, `%`
- `AND`, `OR`, `NOT`
- comparisons `<`, `>`, `<=`, `>=`, `=`, `!=`, `BETWEEN`, `IN` (list-of-constants) as filters
- Boolean-valued expressions (non-filters) using `AND`, `OR`, `NOT`, `<`, `>`, `<=`, `>=`, `=`, `!=`
- `IS [NOT] NULL`
- all math functions (`SIN`, `LOG`, etc.)
- string functions `SUBSTR`, `CONCAT`, `TRIM`, `LTRIM`, `RTRIM`, `LOWER`, `UPPER`, `LENGTH`
- type casts
- Hive user-defined functions, including standard and generic UDFs
- date functions (`YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `UNIX_TIMESTAMP`)
- the IF conditional expression

User-defined functions are supported using a backward compatibility bridge, so although they do run vectorized, they don't run as fast as optimized vector implementations of built-in operators and functions. Vectorized filter operations are evaluated left-to-right, so for best performance, put UDFs on the right in an ANDed list of expressions in the WHERE clause. E.g., use
column1 = 10 and myUDF(column2) = "x"

instead of

myUDF(column2) = "x" and column1 = 10

This will allow the optimized filter to run first, potentially eliminating many rows from consideration, before running the UDF via the bridge. The UDF will only be run for rows that pass the filter on the left hand side of the AND operation.

Using a built-in operator or function that is not supported for vectorization will cause your query to run in standard row-at-a-time mode. If a compile time or run time error occurs that appears related to vectorization, please file a Hive JIRA. To work around such an error, disable vectorization by setting hive.vectorized.execution.enabled to false for the specific query that is failing, to run it in standard mode.

Vectorized support continues to be added for additional functions and expressions. If you have a request for one, please comment on this page, or open a JIRA for it.

Seeing whether vectorization is used for a query

You can verify which parts of your query are being vectorized using the `explain` feature. For example, when Fetch is used in the plan instead of Map, it does not vectorize and the explain output will not include the "Vectorized execution: true" notation:

```sql
create table vectorizedtable(state string,id int) stored as orc;
insert into vectorizedtable values('haryana',1);
set hive.vectorized.execution.enabled = true;
explain select count(*) from vectorizedtable;
```

The `explain` output contains this:

```
STAGE PLANS:
 Stage: Stage-1
  Map Reduce
    Alias -> Map Operator Tree:
      alltypesorc
        TableScan
          alias: vectorizedtable
            Statistics: Num rows: 1 Data size: 95 Basic stats: COMPLETE Column stats: COMPLETE
          Select Operator
            Statistics: Num rows: 1 Data size: 95 Basic stats: COMPLETE Column stats: COMPLETE
          Group By Operator
            aggregations: count()
            mode: hash
            outputColumnNames: _col0
            Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: COMPLETE
          Reduce Output Operator
            sort order:
            Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: COMPLETE
            value expressions: _col0 (type: bigint)
            Execution mode: vectorized
            Reduce Operator Tree:
              Group By Operator
                aggregations: count(VALUE._col0)
                mode: mergepartial
                outputColumnNames: _col0
                Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: COMPLETE
              File Output Operator
                compressed: false
                Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: COMPLETE
              table:
                input format: org.apache.hadoop.mapred.TextInputFormat
                output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
        Execution mode: vectorized
        Reduce Operator Tree:
          Group By Operator
            aggregations: count(VALUE._col0)
            mode: mergepartial
            outputColumnNames: _col0
            Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: COMPLETE
          File Output Operator
            compressed: false
            Statistics: Num rows: 1 Data size: 8 Basic stats: COMPLETE Column stats: COMPLETE
            table:
              input format: org.apache.hadoop.mapred.TextInputFormat
              output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
              serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
```
The notation `Vectorized execution: true` shows that the operator containing that notation is vectorized. Absence of this notation means the operator is not vectorized, and uses the standard row-at-a-time execution path.

Note: In case you want to use vectorized execution for fetch then

```sql
set hive.fetch.task.conversion=none
```

Limitations

- Timestamps only work correctly with vectorized execution if the timestamp value is between 1677-09-20 and 2262-04-11. This limitation is due to the fact that a vectorized timestamp value is stored as a long value representing nanoseconds before/after the Unix Epoch time of 1970-01-01 00:00:00 UTC. Also see HIVE-9862.

Version Information

Vectorized execution is available in Hive 0.13.0 and later (HIVE-5283).