Vectorized Query Execution In Apache Hive

Jitendra Nath Pandey

Hortonworks Inc.

Background

- What is Apache Hive?
  - SQL query engine on Apache Hadoop
  - Map-reduce is the execution engine, Implemented in Java
- Performance concerns for hive
  - Initially designed for batch processing, flexibility
- Multi-pronged solution (Project Stinger at Hortonworks)
  - Better Query Planning
  - New Distributed Execution Engine: TEZ
  - Vectorized Query Processing
  - CPU performance
- This talk’s focus
  - Vectorized Query Processing

Background Contd.

- Row by Row processing in Hive
  - A single row is processed through the entire operator tree before the next row is picked.
  - Inefficient use of CPU instruction pipeline
  - Inefficient use of superscalar (hyper-pipelined) processors
  - Poor cache locality.
    - Layer of object inspectors
    - Run time type inference
    - Lots of virtual method calls, branching in the inner loop
- Result: Low IPC

Vectorized Query Processing

- Process data in a batch of 1024 rows
- Instead of processing a row at a time, process a column vector with 1024 values at a time
  - c = a + b vectorC = vectorA + vectorB
- 1024 has been chosen so that the row batch fits in cache. In most cases L2 cache is enough
- Column vectors are arrays of primitive types, as far as possible
  - Decimal is an exception

Vectorized Query Processing

- Minimize branching in the inner loop, i.e. the loop that processes column vectors of a row batch for an expression.
  - An expression is a unit of work e.g. addition
- Remove the layer of object inspectors.
  - Eager deserialization
  - No type inference at run time
- No object allocations in the inner loop
- Minimal function calls in the inner loop

The Inner Loop

```
Class VectorizedRowBatch {
   ColumnVector [] cols;
   int [] selected;
   boolean selectedInUse;
}
Class DoubleColumnAddDoubleScalar {
      int inputIndex;
      int outputIndex;
      double scalarValue;

     void evaluate (VectorizedRowBatch batch) {
          double [] vector1 = (DoubleColumnVector) batch.cols[inputIndex];
          double [] outputVector = batch.cols[outputIndex];

          if (batch.selectedInUse) {
               for(int j = 0; j != batch.size; j++) {
                     int i = batch.selected[j];
                     outputVector[i] = vector1[i] + scalarValue;
               }
          } else {
               for(int i = 0; i != batch.size; i++) {
                     outputVector[i] = vector1[i] +  scalarValue;
               }
          }
     }
}
```

It works in JAVA too!

- Java Worries
  - Cannot use SIMD (Java 8)
  - Non contiguous arrays.
  - Cache locality could go for a toss
  - Runtime checks
  - Reliance on JIT
- But, the preliminary results were good

Preliminary Evaluation

- CPU performance
  - In memory deserialized data as input
  - Filter operator
  - Select a, b from Table where a = 10;
  - 8x performance improvement
- End to end improvement 3x

Optimizations

- Optimized handling for column vectors with no nulls
- Optimized handling for column vectors with repeating values
- Optimized filters
- Short circuit evaluation
- Vectorized Row Batch is created once and is re-used. All computation is in-place
  - Cache locality

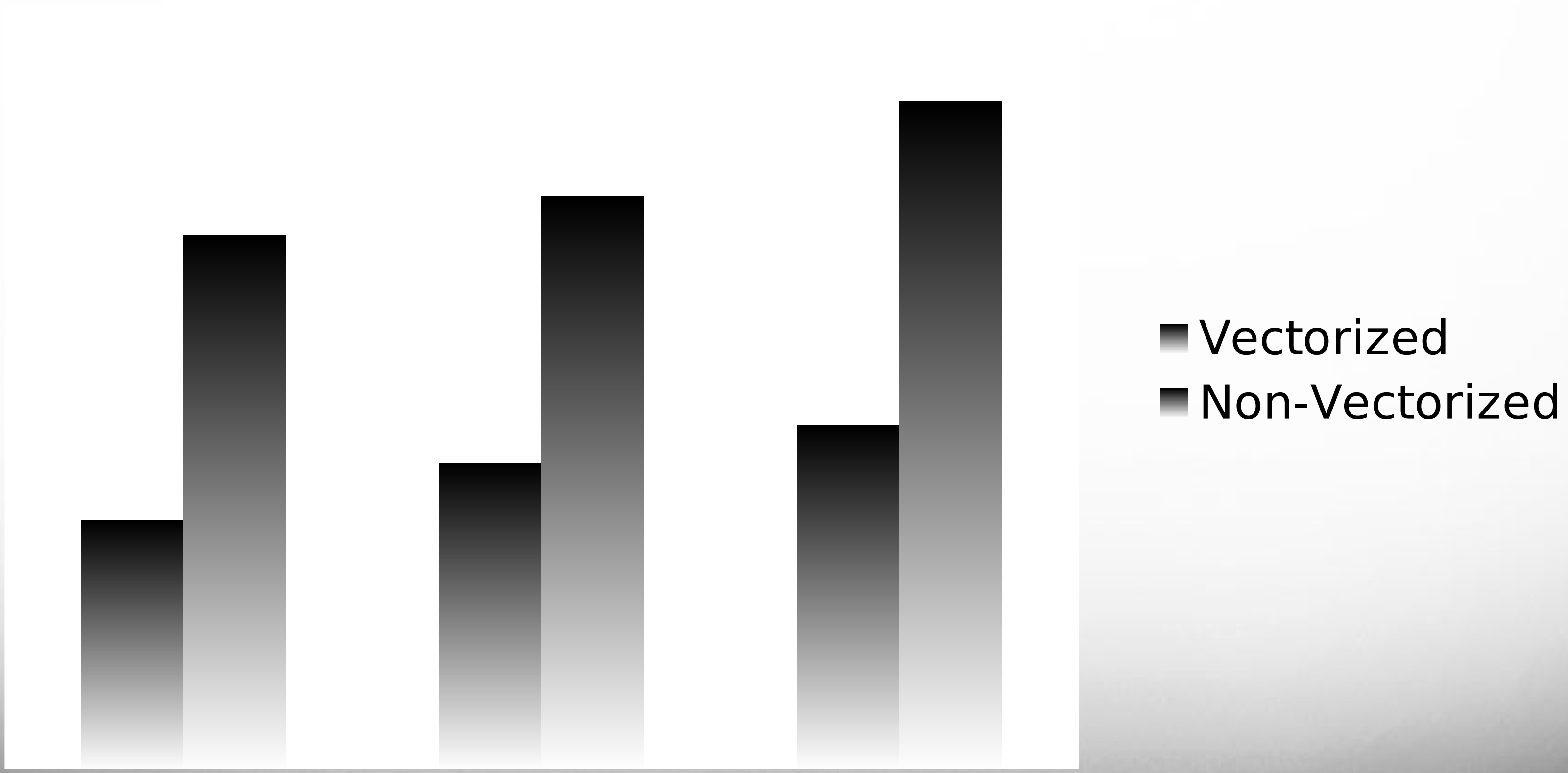
Code Generation

- Type specific code
  - Specialized implementations for the same expression for different data types.
  - LongColAddDoubleScalar
  - DoubleColAddDoubleScalar
- The code is generated from templates
  - Pre-compiled in the code base.
  - The planner puts the right expression class based on the data types involved.
  - No code generation using LLVM (Future work)

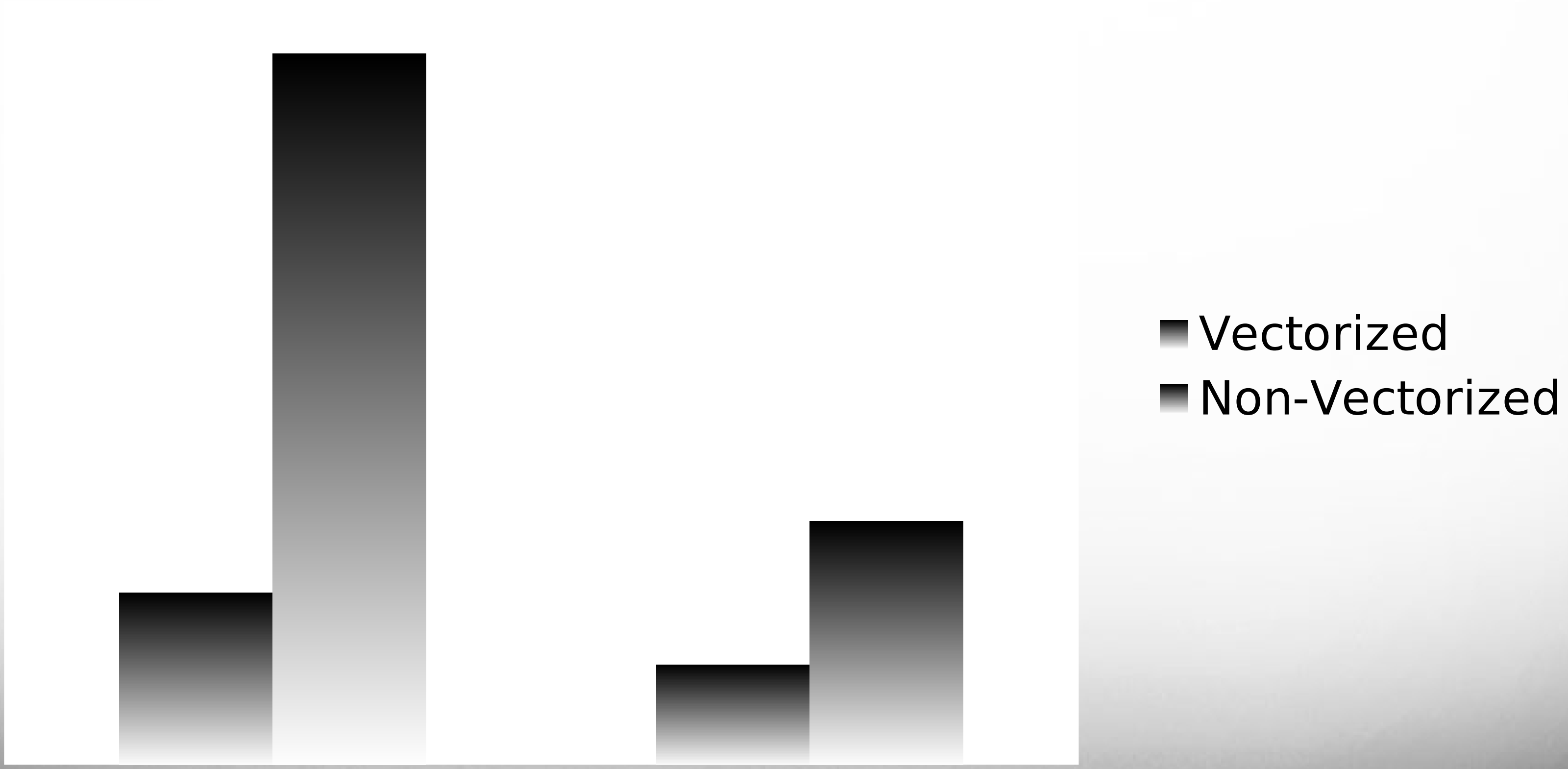
Vectorized Reader

- How is a vectorized row batch loaded?
  - Columnar data formats are more efficient
- ORC Input format
  - https://issues.apache.org/jira/browse/HIVE-3874
- Other file formats
  - Need a vectorized reader
  - One can implement a vectorized reader for any input format
  - Considering to add an adapter layer that buffers up rows into vectorized row batches
  - It is very important to efficiently load the row batches.

Evaluation



TPCH



Try it out

- Vectorization is available in Apache Hive-0.13.
- Enable using
  - SET hive.vectorized.execution.enabled=true;
- The data needs to be in ORC format
  - Other formats will be supported in future releases.

Upcoming Releases

- Reduce side vectorization
  - Shuffle join
  - Windowing functions
  - Multi-staged query processing
- Optimized Join implementations
- More datatypes
  - Varchar
  - Char
  - Complex datatypes
- More optimized decimal implementation.

Thanks!

- A geographically distributed team.
- Started as a joint project between Hortonworks/Microsoft
- Key Contributors
  - Hortonworks
  - Jitendra Pandey, Gopal V, Sergey Selukhin, Teddy Choi
  - Microsoft
  - Eric Hanson, Remus Rusanu, Sarvesh Sakalanaga, Tony Murphy
  - Others
  - Tim Chen, Hideaki Kimura
- Contributors are welcome!

- Contact me at:
  - jitendra@hortonworks.com
  - jitendra@apache.org