

Performance Improvements in Apache Spark 2.0

Whole-stage code generation and vectorization

Markus Dale, Databricks
(<http://tinyurl.com/markus-spark-2-0>)

June 2016

What's in Apache Spark 2.0.0?

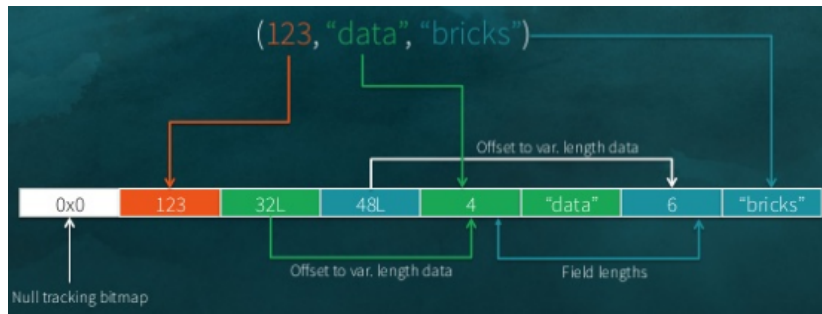
- ▶ Over 2,000 JIRA tickets
- ▶ Some ticket titles:
 - ▶ (Dataset-oriented) API evolution in Spark 2.0
 - ▶ Create a full-fledged built-in SQL parser
 - ▶ Add support for off-heap caching
 - ▶ Model export/import for Pipeline API
 - ▶ Whole stage codegen
 - ▶ Vectorize parquet decoding using ColumnarBatch

Project Tungsten - Closer to bare metal

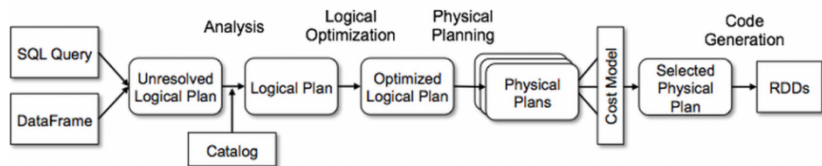
- ▶ Apache Spark 1.3 introduced DataFrames/Catalyst Optimizer
- ▶ Tungsten 1.0:
 - ▶ Memory management and binary processing
 - ▶ Code generation for expression evaluation
 - ▶ See Project Tungsten: Bringing Apache Spark Closer to Bare Metal, Xin and Rosen (2015)

Tungsten Binary Format

► Spark 1.5



Catalyst Optimizer



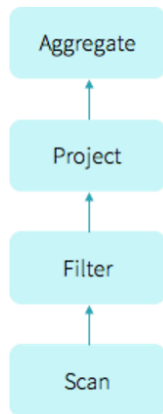
See Databricks Catalyst Optimizer blog entry, Armbrust et al. (2015)

Project Tungsten 2.0 - reduce CPU bottlenecks

- ▶ virtual function calls
- ▶ reading or writing intermediate data to CPU cache or memory

Simple aggregate query with filter

```
select count(*) from store_sales  
where ss_item_sk = 1000
```



Pre-2.0 Apache Spark: Volcano Iterator Model

```
class Filter(child: Operator, predicate: (Row => Boolean))  
  extends Operator {  
    def next(): Row = {  
      var current = child.next()  
      while (current != null && !predicate(current)) {  
        current = child.next()  
      }  
      return current  
    }  
  }  
}
```

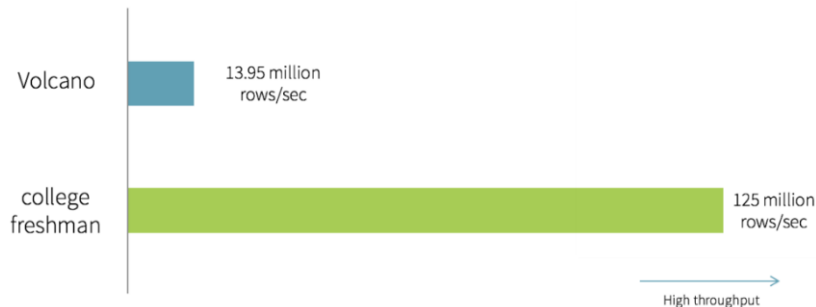
See Graefe (1994)

Handwritten Code

```
var count = 0
for (ss_item_sk in store_sales) {
  if (ss_item_sk == 1000) {
    count += 1
  }
}
```

Handwritten vs. Volcano

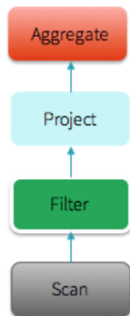
- ▶ Single threaded data from Parquet on disk



Whole-Stage Code Generation Benefits

- ▶ No virtual function dispatches
- ▶ Intermediate data in CPU registers
- ▶ Loop unrolling and SIMD

Whole-Stage Code Generation Example



```
long count = 0;
for (ss_item_sk in store_sales) {
    if (ss_item_sk == 1000) {
        count += 1;
    }
}
```

See Whole-Stage Code Generation with explain()

```
spark.range(1000).  
  filter("id > 100").  
  selectExpr("sum(id)").explain()
```

```
== Physical Plan ==  
*Aggregate(functions=[sum(id#201L)])  
+- Exchange SinglePartition, None  
   +- *Aggregate(functions=[sum(id#201L)])  
      +- *Filter (id#201L > 100)  
         +- *Range 0, 1, 3, 1000, [id#201L]
```

Vectorization

- ▶ Use if unable to do whole-stage codegen
- ▶ Each "next" call runs operator on batched column value

Row Format

1	john	4.1
2	mike	3.5
3	sally	6.4

Column Format

1	2	3
john	mike	sally
4.1	3.5	6.4

Demo

- ▶ SparkSession
- ▶ 1.6 vs. 2.0 TSV file
- ▶ ETL to Parquet
- ▶ 1.6 vs. 2.0 on Parquet file
- ▶ See <http://tinyurl.com/markus-spark-2-0> and then src (or <https://github.com/medale/presentations/tree/master/spark-performance-2.0-2016-06/src>) for notebooks
- ▶ <https://databricks.com/try-databricks> - Databricks Community Edition

References I

Armbrust, Michael, Yin Huai, Cheng Liang, Reynold S. Xin, and Matei Zaharia. 2015. "Deep Dive into Spark SQL's Catalyst Optimizer." <https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html>.

Graefe, G. 1994. "Volcano-an Extensible and Parallel Query Evaluation System." *IEEE Transactions on Knowledge and Data Engineering* 6 (1) (February): 120–135. doi:10.1109/69.273032.

Xin, Reynold S., and Josh Rosen. 2015. "Project Tungsten - Bringing Apache Spark Closer to Bare Metal." <https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html>.