SQL-on-Hadoop

Full Circle back to Shared-Nothing Database Architectures

Problem

- Compare the performance of two SQL-on-Hadoop systems: Hive vs Impala
  - Through experiments and analysis that highlights their different design tradeoffs and the impact on performance

Why is it important?

- Many different SQL-on-Hadoop systems
  - Two general categories: Native Hadoop-based and database-Hadoop hybrids
  - Hive and Impala are both Native Hadoop-based
  - The hybrids combine Hadoop scheduling and fault-tolerance with (Postgre)SQL
- Architectural differences
  - Hive: uses a run-time framework, such as MapReduce or Tez, for scheduling, data movement and parallelization
  - Impala: emerging SQL-on-Hadoop systems, which follow a shared-nothing database like architecture
  - good representations of their relevant categories, popular, and widely used in the enterprise

Background: Apache Hive

- SQL-like query language: HiveQL
- HiveQL ➔ Query Execution Plan (Directed Acyclic Graph of MapReduce tasks) ➔ Executed via framework
- Optimizations
    - pipelines data through execution stages in Tez in latest versions, but still bound by Java deserialization

Background: Cloudera Impala

- Has a SQL-like query language, too
- Provides its own long running daemon on each node of the cluster
- Shared-nothing parallel database architecture
- Optimizations
  - highly efficient I/O layer to read data stored in HDFS
  - exploit streaming SIMD extension (SSE)
- Disadvantage
  - Requires the working set of a query to fit in the aggregate physical memory of the cluster it runs on.

Columnar File Format: Hive’s ORC

- Optimized Row Columnar: storage efficiency by providing data encoding, block level compression, better IO via lightweight built-in index
- Stores multiple groups of row data as stripes.
- Each file has a file footer that stores metadata and aggregates such as count, sum, max, min.
    - These can be used to skip stripes for queries.

Columnar File Format: Impala's Parquet

- Designed to exploit efficient compression and encoding schemes, and to support nested data
- Data Hierarchy
  - Stores data grouped together in logical horizontal partitions called row groups.
  - Every row group contains a column chunk for each column in the table.
  - Each column chunk consists of multiple pages and is guaranteed to be stored contiguously on disk
- Metadata is stored at each level of the hierarchy, and can be used to skip queries

Experiments

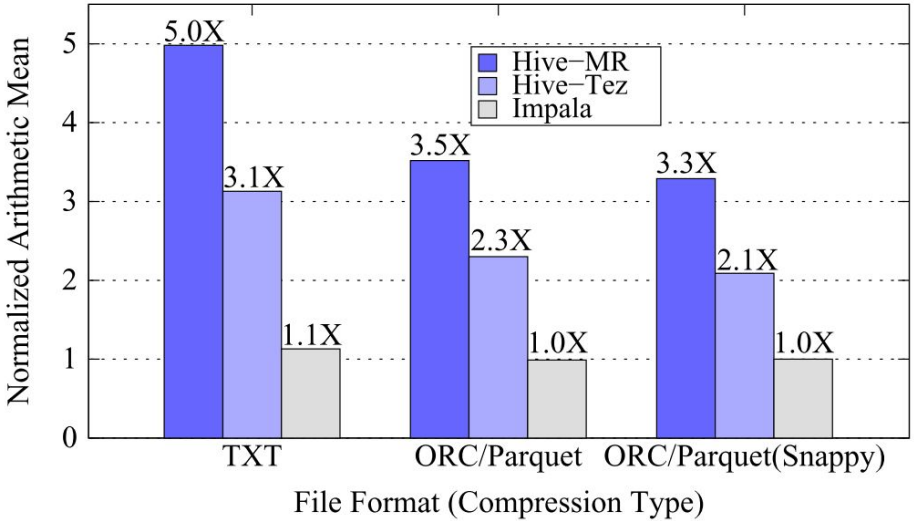
- Setup
  - 21 node cluster (20 compute nodes)
  - 12 cores per node
- 4 set of tests
  - TPC-H like workload
  - 2 TPC-DS inspired workloads
  - Micro-Benchmarks
    - Used to compare performance in each system
  - ORC Indexing Test

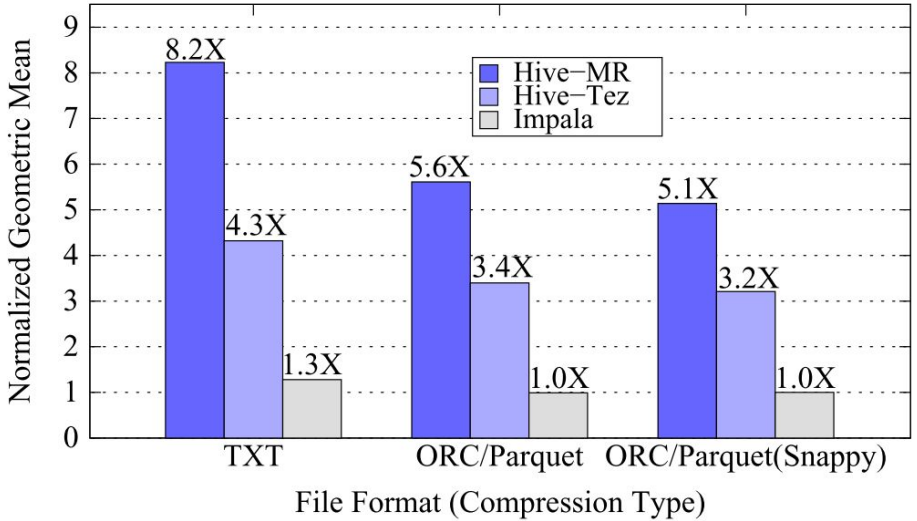
TPC-H workload

- 22 read-only TPC-H queries. scale factor of 1000 GB
  - scale factor limited by Impala because the workload's working set has to fit in cluster aggregate memory
- 11 queries rewritten to take advantage of system features on Hive
- 3 storage format for each system are tested
  - Text, ORC/Parquet, ORC/Parquet + compression
- 2 queries failed to complete
  - 1 due to lack of sufficient memory in Impala
  - 1 was killed after 2000 seconds in Hive

TPC-H Results

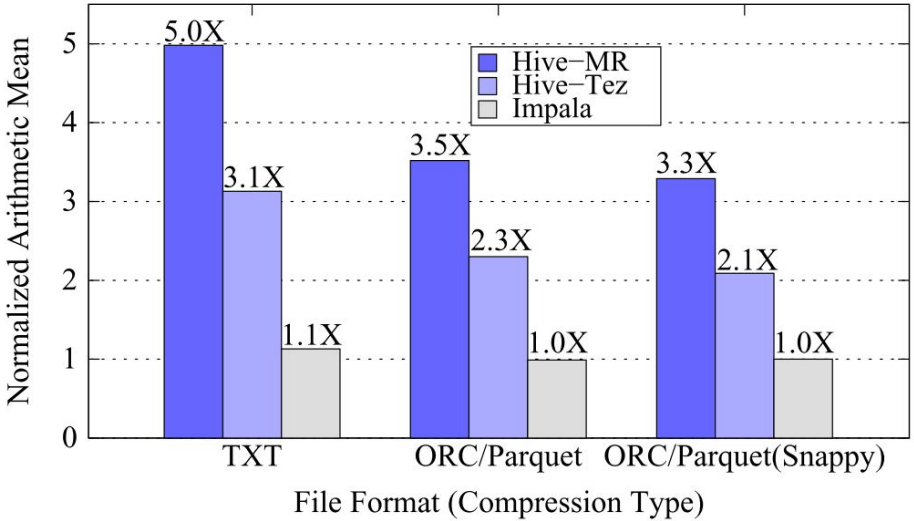
- Impala is faster than Hive-MR and Hive-Tez

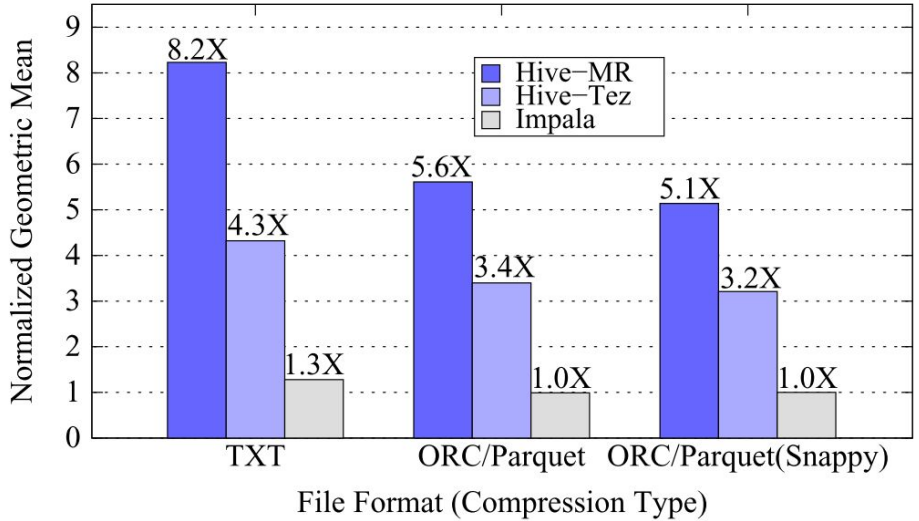




TPC-H Results

- Impala does not improve when moving from TXT to Parquet
  - not I/O bound





Why Impala is faster

- Impala
  - More efficient I/O subsystem
  - Parquet files are generally smaller than ORC files
  - Impala is able to generate code at runtime
    - Eliminate overheads of virtual function calls, inefficient instruction branching, etc
    - 1.3x speedup
  - Impala query is pipelined, whereas Hive-MR has to enforce data materialization at every step
  - (-) Aggregation operations are single-threaded
- Hive
  - Java deserialization overheads becomes bottlenecks
  - MapReduce in Hive-MR has overhead of starting multiple map tasks when reading a large file
    - Hive-Tez and Impala avoids the startup overheads

TPC-DS inspired workloads

- 20 queries
  - Access 1 single fact table and 6 dimension tables
- Published by Impala developers. Authors try to reproduce them
  - May be biased towards Impala?
- Two workloads
  - In TPC-DS benchmark, the value of filtering predicates change with the scale factor
  - Developer’s benchmark have filtering predicates that do not match TPC-DS specification

Results: TPC-DS 1

- Impala is on average 8.2x faster than Hive MR and 4.3X faster than Hive-Tez

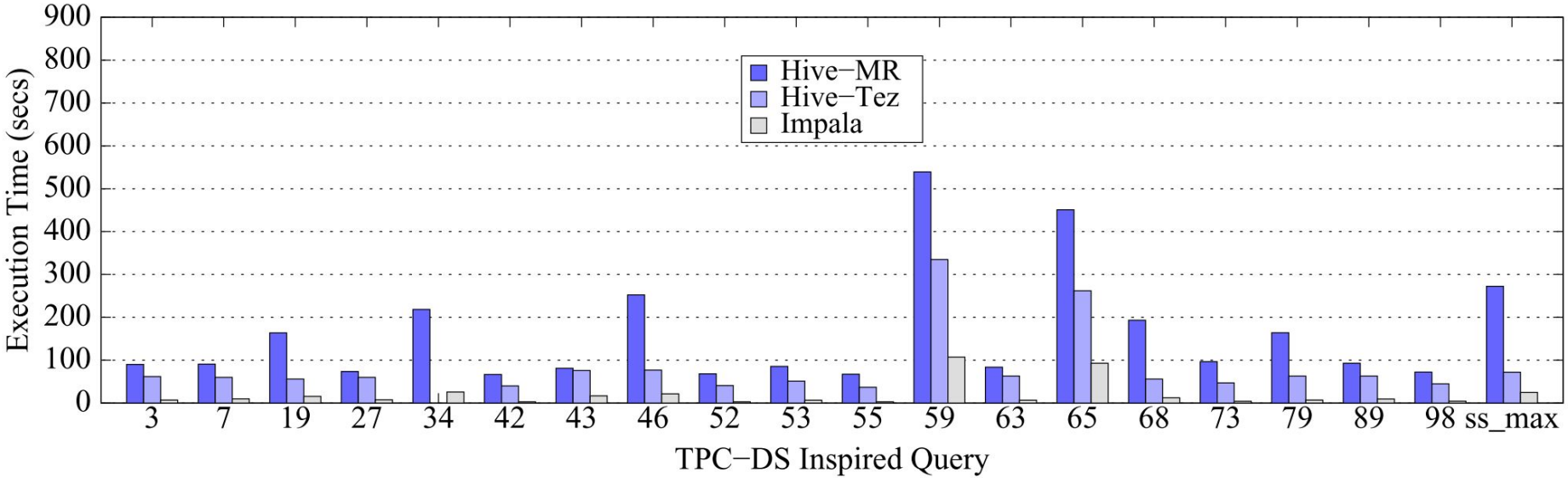


Figure 6: TPC-DS inspired Workload 1

Results: TPC-DS 2

- Impala is 10x faster than Hive-MR and 4.4X faster than Hive-Tez

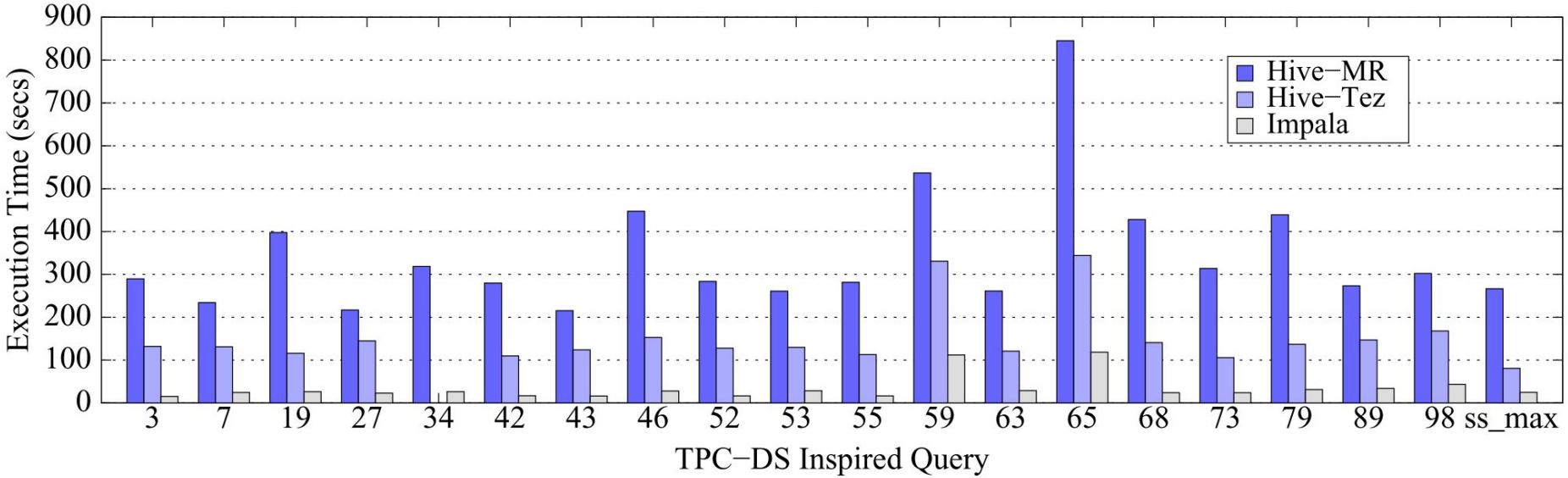


Figure 7: TPC-DS inspired Workload 2

TPC-DS Analysis

- Impala’s advantage for workloads that access one fact table and multiple small dimension tables
- Impala has an efficient I/O system. The Hive-MR has significant overheads when ingesting small tables
- Difference between Hive-Tez and Hive MR is because Hive-Tez avoids Hive-MR’s scheduling and materialization overheads

Micro Benchmarks

- Study the I/O characteristics of Hive-MR and Impala over their storage formats
- Comparison within each system. Not across
- Data: 3 data files of 200 GB each, with varying number of integer columns (10, 100, 1000)
- Each experiment scan a fixed percentage of the columns and record the read throughput, divided by the scan time
- Tests were run with both sequential access and random access

Micro Benchmark: Impala

- Sequential Access
  - Read throughput for 100 and 1000 columns remain almost constant
  - Throughput on the 1000-column is lower than that of the 100-column
  - Read throughput on the 10-column dataset increases as the number of accessed columns increases
    - bottleneck: tuple materialization

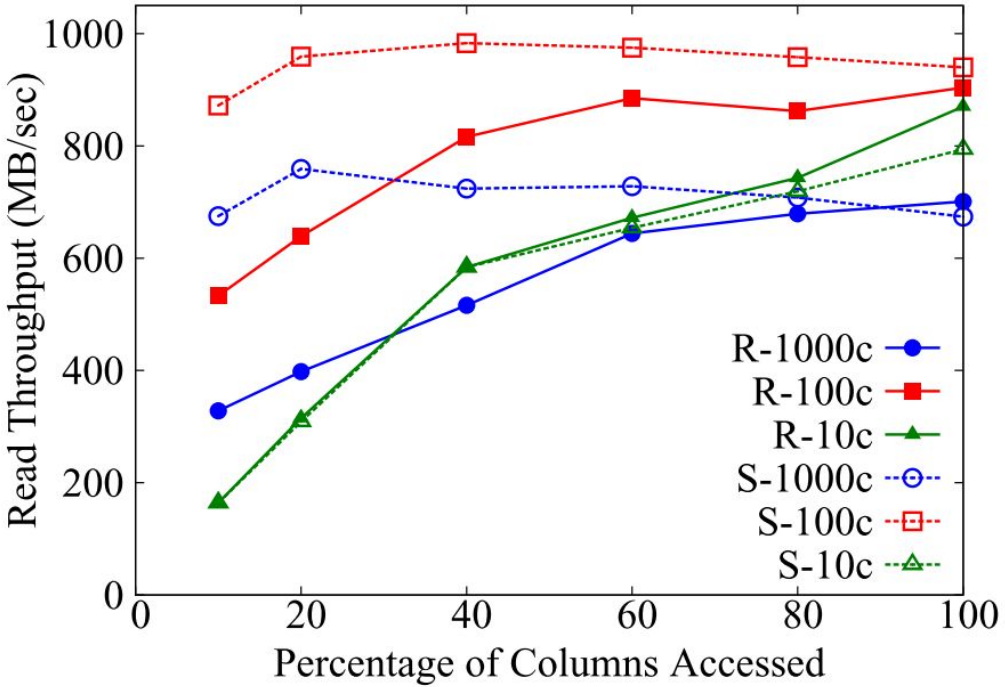


Figure 8: Impala/Parquet Read Throughput

Micro Benchmark: Impala

- Random access
  - read throughput increases as the number of accessed columns increases

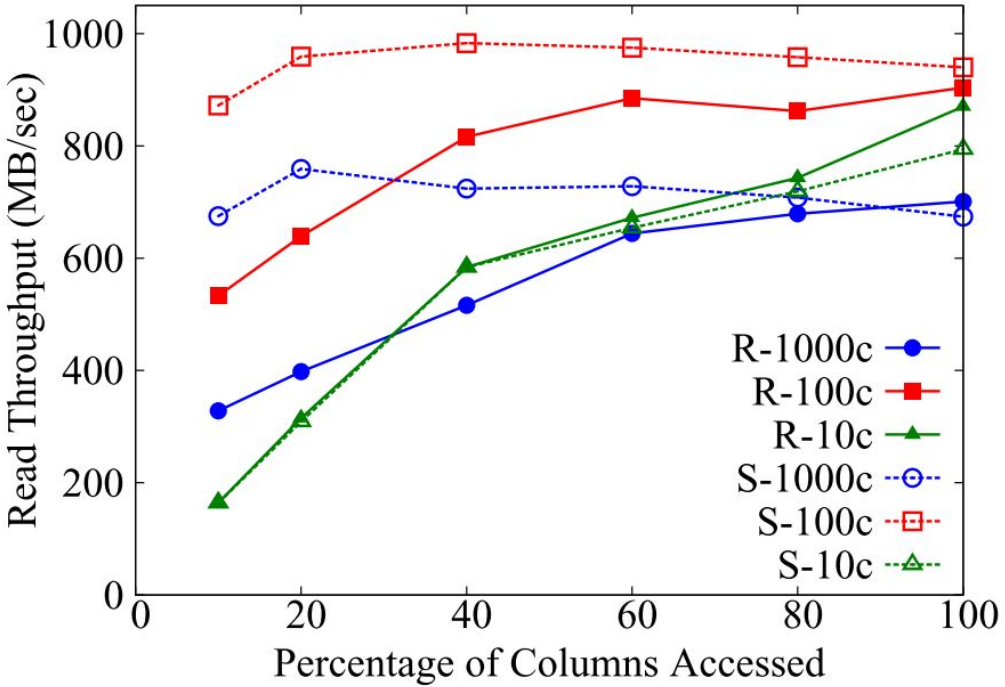


Figure 8: Impala/Parquet Read Throughput

Micro Benchmark: Hive

- Read throughput always increases as the number of accessed columns increases
  - High overheads of starting and stopping Map tasks
- CPU utilization is 87%. Disks are under-utilized (< 30%)
  - object deserialization overhead

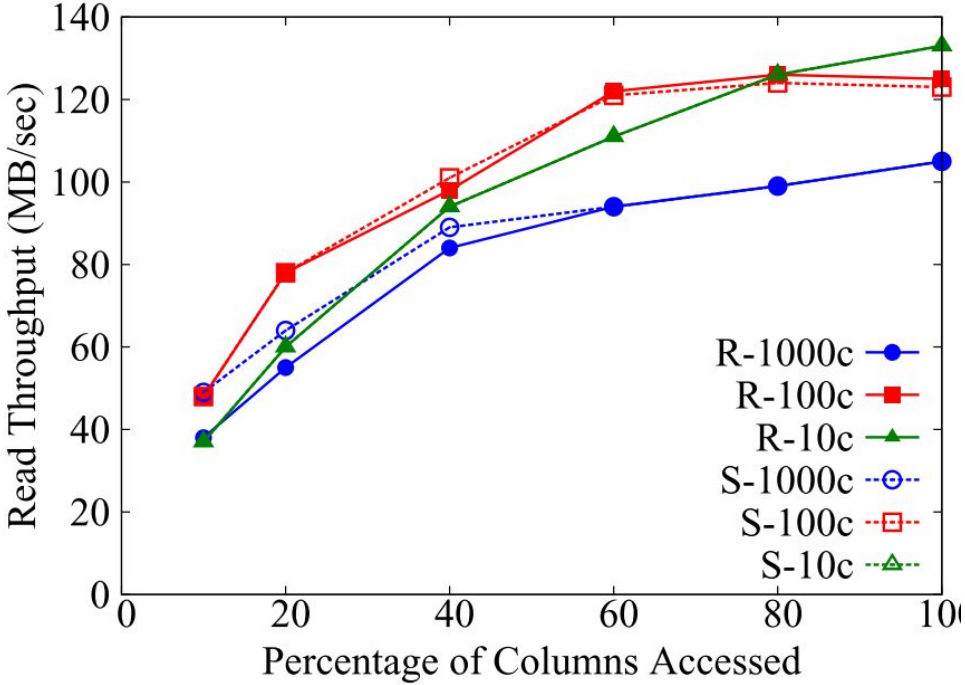


Figure 9: Hive/ORC Read Throughput

ORC Indexing

- ORC uses a coarse-grained indexing stored in a stripe
- Experiment: 2 10-column dataset, one sorted, one unsorted
- Index not helpful for unsorted dataset

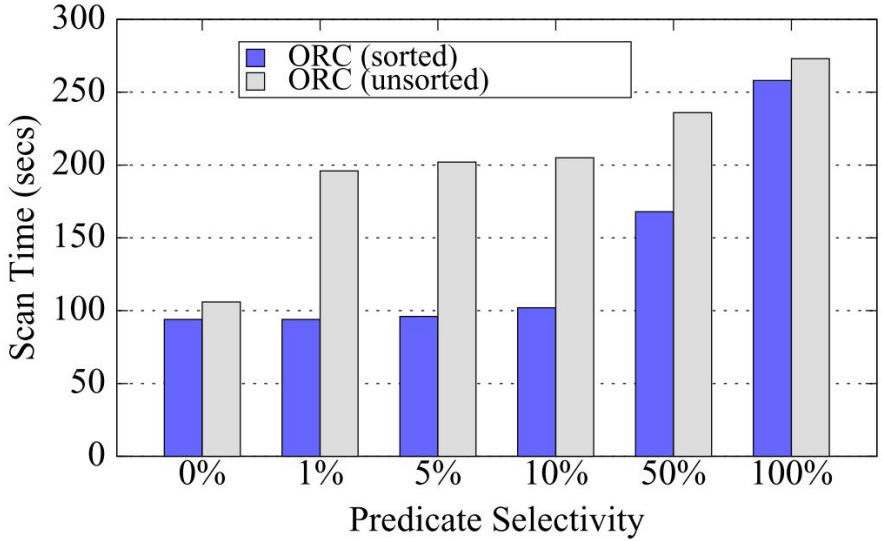


Figure 10: Predicate Selectivity vs. Scan Time

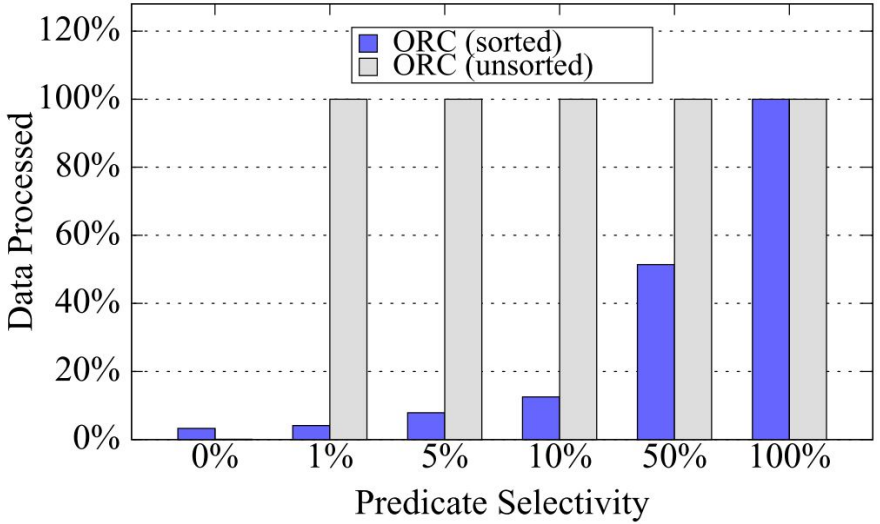


Figure 11: Predicate Selectivity vs. Data Processed

Future Works

- Comparison with other types of SQL-on-Hadoop systems
  - Recall the two general categories: Native Hadoop-based and database-Hadoop hybrids
- Scaling up to more nodes
  - All tests are now on 21 nodes