Impala

Eric Tu

What is Impala?

- Run fast queries on top of Hadoop ecosystem
- SQL on Hadoop: Plug into existing systems, no need to relearn
- Hive was SQL on Hadoop, translated SQL into MapReduce jobs
- Distributed SQL Engine for Hadoop
  - Bypasses MapReduce
  - Uses different parts of the Hadoop ecosystem (HDFS, HBase, YARN)
  - Reads Hadoop file formats (Parquet, RC)
- Low latency, high concurrency for read-mostly queries
  - BI focused, analytic, short queries

SQL in Impala

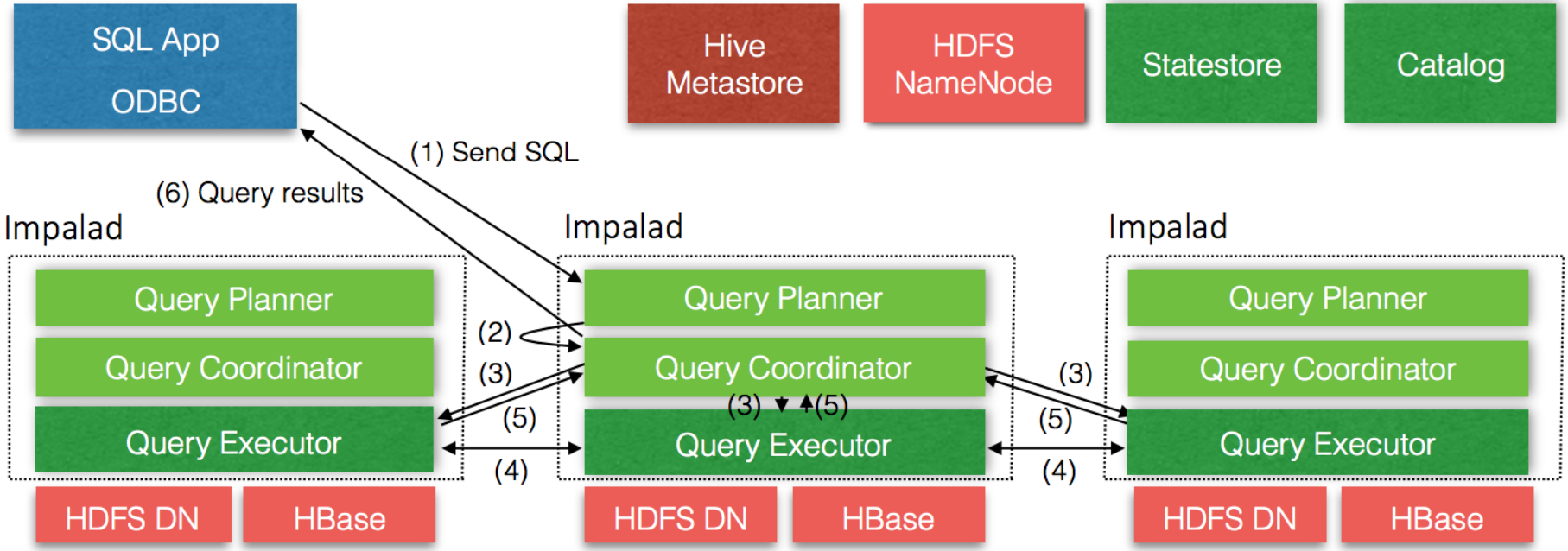
- No Update/Delete
  - Limitations of HDFS, so only bulk insertions
- Add data to table by copying/moving files into HDFS directory of table
- When creating table, specify file format and placement
- User needs to run COMPUTE STATS for query optimization

General Architecture

- Impalad
  - Daemon service that both accepts queries and orchestrates execution
- Statestore
  - Updates metadata to Impala processes
- Catalog
  - Metadata access via statestore broadcasting
  - Tables, views, columns, files, block replica location, etc.
  - Plug in to existing metastores: Hive Metastore

Impalad

- 3 Segments: Planner, Coordinator, Executor
- Planner turns request into plan fragments
- Can have dual role of Planner/Coordinator:
  - Accepts Queries and orchestrates execution
  - Executes Query Fragments from other Impalads
  - Dual Role -> fault-tolerance, load balancing
- One per data node in cluster
  - Data locality since HDFS remote reads are expensive

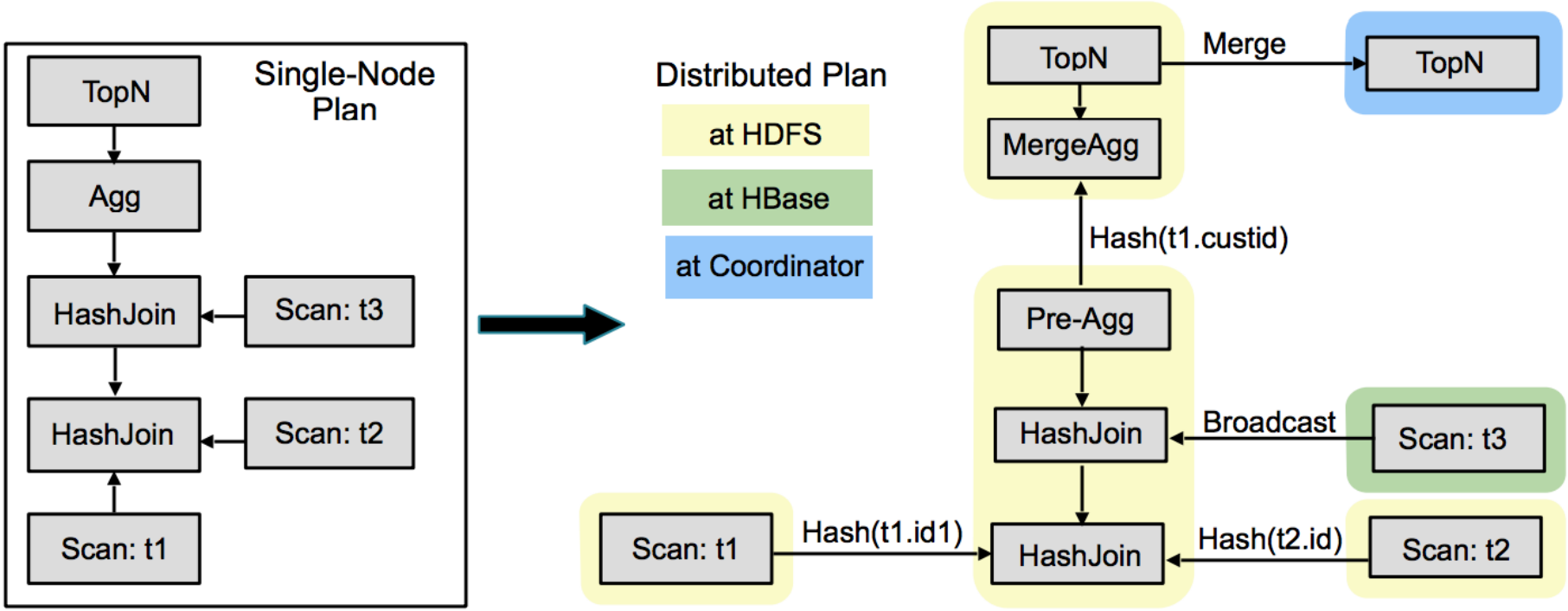

SQL App
ODBC
Hive Metastore
HDFS NameNode
Statestore
Catalog
(1) Send SQL
(6) Query results
Impalad
Impalad
Impalad
Query Planner
Query Coordinator
Query Executor
Query Planner
Query Coordinator
Query Executor
Query Planner
Query Coordinator
Query Executor
(2)
(3)
(5)
(4)
(3)
(5)
(3)
(5)
(4)
HDFS DN
HBase
HDFS DN
HBase
HDFS DN
HBase

Statestore

- Single Instance
- **“Avoid synchronous RPCs whenever possible”**
- Topic: Array[(key, value, version)]
- Subscribe
  - Processes register with statestore for interest in topic
- Push
  - Topic update: delta updates for topic
  - Keepalive: detect timeouts for subscription
- No coordination
  - Information required to execute a plan is distributed to that node

Frontend: Compiling SQL into query plans

- General Goal: **Data Locality**
- Single node planning
  - Given parse tree -> single node plan tree
  - Assign predicates at lowest possible plan node
- Plan parallelization and fragmentation
  - Add more plan nodes for local aggregation, data exchange
  - **Joins:** Choose broadcast or partitioned to minimize data exchange
  - **Aggregations:** preaggregation (materialize data) -> merge aggregation on grouping column
  - **Plan Fragments:** portion of tree operating on the same data partition of a machine


Single-Node Plan
TopN
Agg
HashJoin
Scan: t3
HashJoin
Scan: t2
Scan: t1
Distributed Plan
at HDFS
at HBase
at Coordinator
TopN
Merge
TopN
MergeAgg
Hash(t1.custid)
Pre-Agg
HashJoin
Broadcast
Scan: t3
Scan: t1
Hash(t1.id1)
HashJoin
Hash(t2.id)
Scan: t2

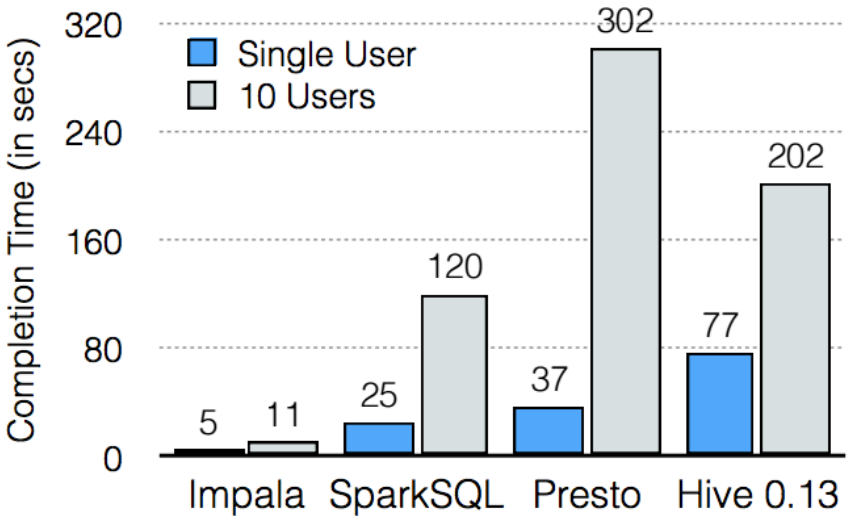
Backend: Running query fragments

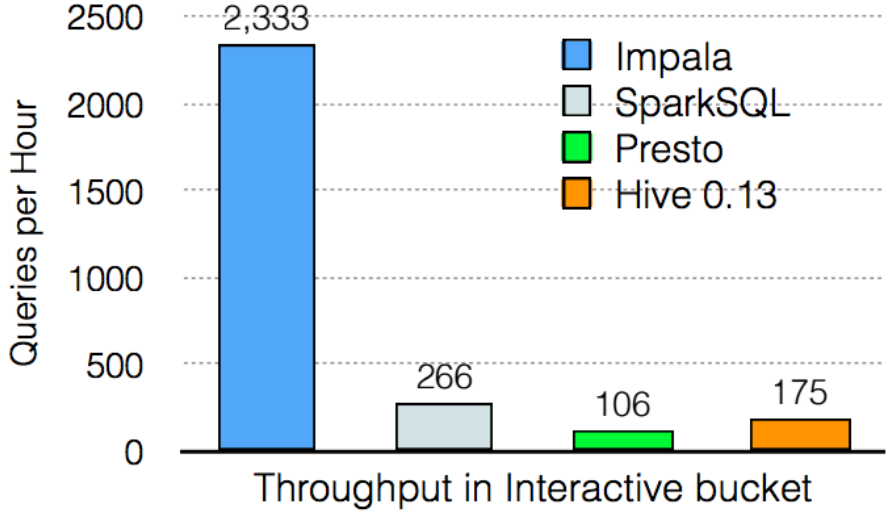
- Pipelined execution
- LLVM- based code generation: inner loop functions
- HDFS features
  - **Goal**: Make data scans close to hardware speed
  - Short-circuit local reads bypass overhead
  - HDFS caching accesses data in memory

Resource Management

- YARN latency with resource acquisition too long
- LLAMA:
  - Services Impalad requests, each associated with resource pool
  - Resource caching can serve resources immediately
  - Adjust resource consumption estimates during query execution
  - Otherwise give request to YARN

Performance





Discussion

- When will traditional DBMS systems phase out?
- Is this modular Hadoop ecosystem the future? Is decoupling from storage good?
- Why not improve/build on top of Hive?