cloudera
Ask Bigger Questions

Cloudera Impala:
A Modern SQL Engine for Apache Hadoop

Marcel Kornacker

Software Engineer, Architect of Cloudera Impala

January 10, 2013

Welcome to the webinar!

- All lines are muted
- Q&A after the presentation
- Ask questions at any time by typing them in the “Questions” pane on your GoToWebinar panel
- Recording of this webinar will be available on demand at cloudera.com

Speaker Bio: Marcel Kornacker

Marcel Kornacker is a tech lead at Cloudera for new products development and creator of the Cloudera Impala project.

Following his graduation in 2000 with a PhD in databases from UC Berkeley, he held engineering positions at several database-related start-up companies. Marcel joined Google in 2003 where he worked on several ads serving and storage infrastructure projects, then became tech lead for the distributed query engine component of Google's F1 project.

Impala Overview: Goals

- General-purpose SQL query engine:
  - should work both for analytical and transactional workloads
  - will support queries that take from milliseconds to hours
- Runs directly within Hadoop:
  - reads widely used Hadoop file formats
  - talks to widely used Hadoop storage managers
  - runs on same nodes that run Hadoop processes
- High performance:
  - C++ instead of Java
  - runtime code generation
  - completely new execution engine that doesn't build on MapReduce

User View of Impala: Overview

- Runs as a distributed service in cluster: one Impala daemon on each node with data
- User submits query via ODBC/Beeswax Thrift API to any of the daemons
- Query is distributed to all nodes with relevant data
- If any node fails, the query fails
- Impala uses Hive's metadata interface, connects to Hive's metastore
- Supported file formats:
  - text files (GA: with compression, including lzo)
  - sequence files with snappy/gzip compression
  - GA: Avro data files
  - GA: Trevni (columnar format; more on that later)

User View of Impala: SQL

- SQL support:
  - patterned after Hive's version of SQL
  - limited to Select, Project, Join, Union, Subqueries, Aggregation and Insert
  - only equi-joins; no non-equi joins, no cross products
  - Order By only with Limit
  - GA: DDL support (CREATE, ALTER)
- Functional limitations:
  - no custom UDFs, file formats, SerDes
  - no beyond SQL (buckets, samples, transforms, arrays, structs, maps, xpath, json)
  - only hash joins; joined table has to fit in memory:
    - beta: of single node
    - GA: aggregate memory of all (executing) nodes

User View of Impala: Apache HBase

- HBase functionality:
  - uses Hive's mapping of HBase table into metastore table
  - predicates on rowkey columns are mapped into start/stop row
  - predicates on other columns are mapped into SingleColumnValueFilters
- HBase functional limitations:
  - no nested-loop joins
  - all data stored as text

Impala Architecture

- Two binaries: impalad and statestored
- Impala daemon (impalad)
  - handles client requests and all internal requests related to query execution
  - exports Thrift services for these two roles
- State store daemon (statestored)
  - provides name service and metadata distribution
  - also exports a Thrift service

Impala Architecture

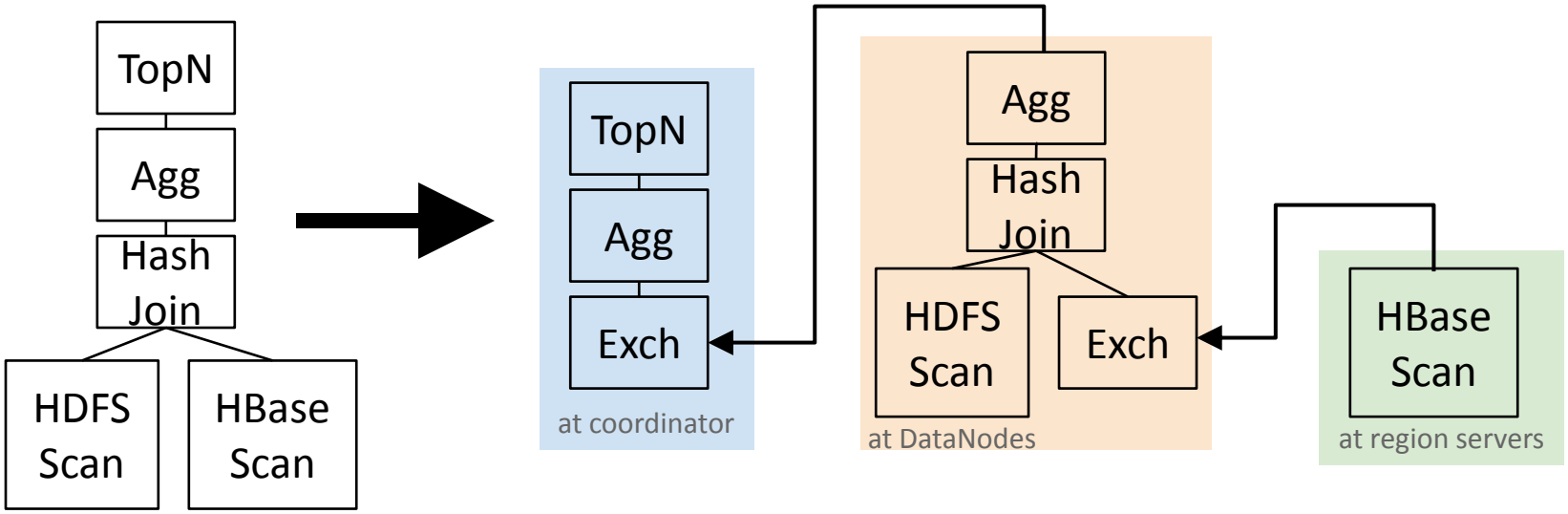
- Query execution phases
  - request arrives via odbc/beeswax Thrift API
  - planner turns request into collections of plan fragments
  - coordinator initiates execution on remote impalad's
  - during execution
    - intermediate results are streamed between executors
    - query results are streamed back to client
    - subject to limitations imposed to blocking operators (top-n, aggregation)

Impala Architecture: Planner

- join order = FROM clause order GA target: rudimentary cost-based optimizer
- 2-phase planning process:
  - single-node plan: left-deep tree of plan operators
  - plan partitioning: partition single-node plan to maximize scan locality, minimize data movement
- plan operators: Scan, HashJoin, HashAggregation, Union, TopN, Exchange
- distributed aggregation: pre-aggregation in all nodes, merge aggregation in single node. GA: hash-partitioned aggregation: re-partition aggregation input on grouping columns in order to reduce per-node memory requirement

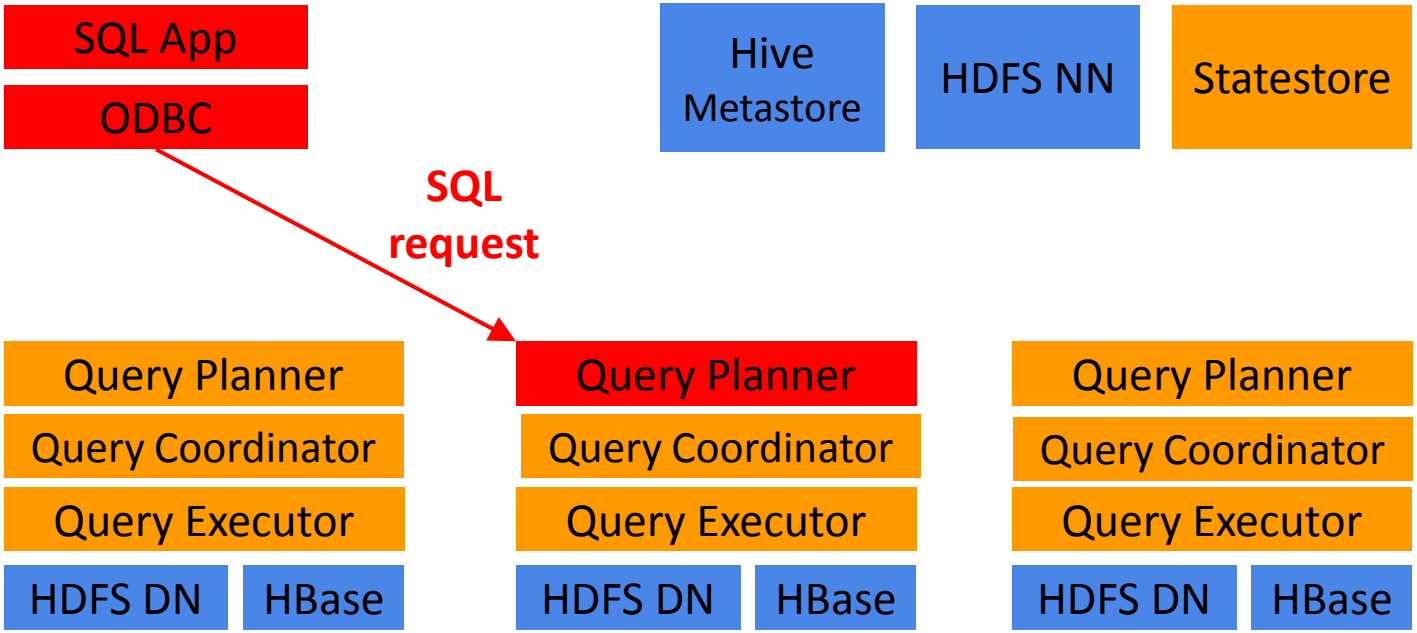
Impala Architecture: Planner

- Example: query with join and aggregation
  SELECT state, SUM(revenue)
  FROM HdfsTbl h JOIN HbaseTbl b ON (...)
  GROUP BY 1 ORDER BY 2 desc LIMIT 10



Impala Architecture: Query Execution

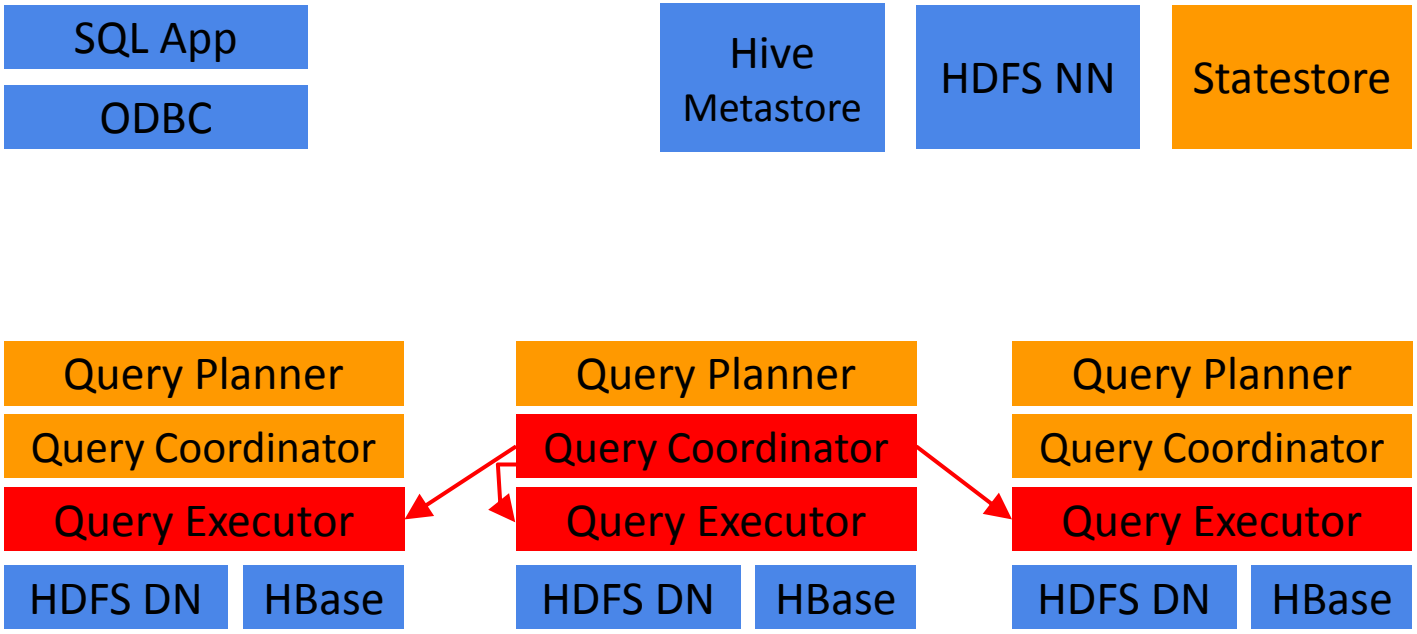
Request arrives via odbc/beeswax Thrift API



Impala Architecture: Query Execution

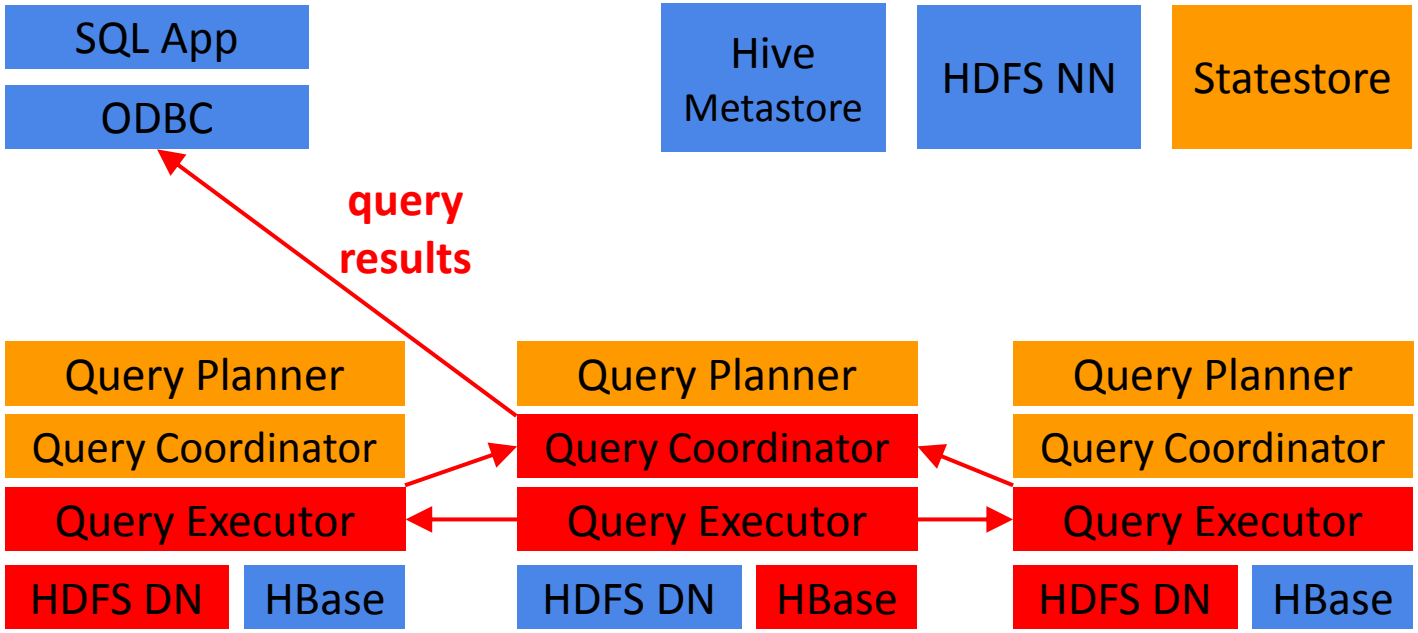
Planner turns request into collections of plan fragments

Coordinator initiates execution on remote impalad's



Impala Architecture: Query Execution

Intermediate results are streamed between impalad's Query results are streamed back to client



Impala Architecture

- Metadata handling:
  - utilizes Hive's metastore
  - caches metadata: no synchronous metastore API calls during query execution
  - beta: impalad's read metadata from metastore at startup
  - GA: metadata distribution through statestore
  - post-GA: HCatalog and AccessServer

Impala Architecture

- Execution engine
  - written in C++
  - runtime code generation for "big loops"
    - example: insert batch of rows into hash table
    - code generation with llvm
    - inlines all expressions; no function calls inside loop
  - all data is copied into canonical in-memory tuple format; all fixed-width data is located at fixed offsets
  - uses intrinsics/special cpu instructions for text parsing, crc32 computation, etc.

Impala's Statestore

- Central system state repository
  - name service (membership)
  - GA: metadata
  - GA: other scheduling-relevant or diagnostic state
- Soft-state
  - all impalad's register at startup
  - impalad's re-register after losing connection
  - Impala service continues to function in absence of statestored (but: with increasingly stale state)
  - State pushed to impalad's periodically
  - Repeated failed heartbeat means impalad evicted from cluster view
- Thrift API for service / subscription registration

Statestore: Why not ZooKeeper?

- ZK is not a good pub-sub system
  - Watch API is awkward and requires a lot of client logic
  - multiple round-trips required to get data for changes to node's children
  - push model is more natural for our use case
- Don't need all the guarantees ZK provides:
  - serializability
  - persistence
  - prefer to avoid complexity where possible
- ZK is bad at the things we care about and good at the things we don't

Comparing Impala to Dremel

- What is Dremel?
  - columnar storage for data with nested structures
  - distributed scalable aggregation on top of that
- Columnar storage in Hadoop: joint project between Cloudera and Twitter
  - new columnar format, derived from Doug Cutting's Trevni
  - stores data in appropriate native/binary types
  - can also store nested structures similar to Dremel's ColumnIO
- Distributed aggregation: Impala
- Impala plus columnar format: a superset of the published version of Dremel (which didn't support joins)

Comparing Impala to Hive

- Hive: MapReduce as an execution engine
  - High latency, low throughput queries
  - Fault-tolerance model based on MapReduce's on-disk checkpointing; materializes all intermediate results
  - Java runtime allows for easy late-binding of functionality: file formats and UDFs.
  - Extensive layering imposes high runtime overhead
- Impala:
  - direct, process-to-process data exchange
  - no fault tolerance
  - an execution engine designed for low runtime overhead

Comparing Impala to Hive

- Impala's performance advantage over Hive: no hard numbers, but
  - Impala can get full disk throughput (~100MB/sec/disk); I/O-bound workloads often faster by 3-4x
  - queries that require multiple map-reduce phases in Hive see a higher speedup
  - queries that run against in-memory data see a higher speedup (observed up to 100x)

Impala Roadmap: GA – Q2 2013

- New data formats:
  - lzo-compressed text
  - Avro
  - columnar
- Better metadata handling:
  - automatic metadata distribution through statestore
- Connectivity: jdbc
- Improved query execution: partitioned joins
- Further performance improvements

Impala Roadmap: GA – Q2 2013

- Guidelines for production deployment:
  - load balancing across impalad's
  - resource isolation within MR cluster
- Additional packages: RHEL 5.7, Ubuntu, Debian

Impala Roadmap: 2013

- Improved HBase support:
  - composite keys, Avro data in columns, index nested-loop joins, INSERT/UPDATE/DELETE
- Additional SQL:
  - UDFs
  - SQL authorization and DDL
  - ORDER BY without LIMIT
  - window functions
  - support for structured data types

Impala Roadmap: 2013

- Runtime optimizations:
  - straggler handling
  - join order optimization
  - improved cache management
  - data collocation for improved join performance
- Resource management:
  - cluster-wide quotas
  - Teradata-style policies ("user x can never have more than 5 concurrent queries running", etc.)
  - goal: run exploratory and production workloads in same cluster, against same data, w/o impacting production jobs

Questions

- Type it in the “Questions” panel
- Recording will be available on demand at cloudera.com
- After webinar, inquire at impala-user@cloudera.org
- My email:

  marcel@cloudera.com

Thank you for attending!

Download Impala (beta)

cloudera.com/downloads

Learn more about Cloudera Enterprise RTQ, Powered by Impala

cloudera.com/impala

Join the Impala Hackathon, Monday, Feb 25 @ Cloudera HQ in Palo Alto

Register: bit.ly/DataHackDay