From Raw Data to Analytics with No ETL

Marcel Kornacker // Cloudera, Inc.
Lenni Kuff // Cloudera, Inc.
Outline

• Evolution of ETL in the context of analytics
  • traditional systems
  • Hadoop today
• Cloudera’s vision for ETL: no ETL
  • with qualifications
Traditional ETL

• Extract: physical extraction from source data store
  • could be an RDBMS acting as an operational data store
  • or log data materialized as json
• Transform:
  • data cleansing and standardization
  • conversion of naturally complex/nested data into a flat relational schema
• Load: the targeted analytic DBMS converts the transformed data into its binary format (typically columnar)
Traditional ETL

• Three aspects to the traditional ETL process:
  1. semantic transformation such as data standardization/cleansing
     -> makes data more queryable, adds value
  2. representational transformation: from source to target schema
     (from complex/nested to flat relational)
     -> “lateral” transformation that doesn’t change semantics,
        adds operational overhead
  3. data movement: from source to staging area to target system
     -> adds yet more operational overhead
Traditional ETL

• The goals of “analytics with no ETL”:
  • simplify aspect 1
  • eliminate aspects 2 and 3
ETL with Hadoop Today

• A typical ETL workflow with Hadoop looks like this:
  • raw source data initially lands in HDFS (examples: text/xml/json log files)
  • that data is mapped into a table to make it queryable:
    CREATE TABLE RawLogData (...) ROW FORMAT DELIMITED FIELDS
    LOCATION '/raw-log-data/';
  • the target table is mapped to a different location:
    CREATE TABLE LogData (...) STORED AS PARQUET LOCATION '/log-data/';
  • the raw source data is converted to the target format:
    INSERT INTO LogData SELECT * FROM RawLogData;
  • the data is then available for batch reporting/analytics (via Impala, Hive, Pig, Spark) or interactive analytics (via Impala, Search)
ETL with Hadoop Today

- Compared to traditional ETL, this has several advantages:
  - Hadoop acts as a centralized location for all data: raw source data lives side by side with the transformed data
  - data does not need to be moved between multiple platforms/clusters
  - data in the raw source format is queryable as soon as it lands, although at reduced performance, compared to an optimized columnar data format
  - all data transformations are expressed through the same platform and can reference any of the Hadoop-resident data sources (and more)
ETL with Hadoop Today

• However, even this still has drawbacks:
  • new data needs to be loaded periodically into the target table, and doing that reliably and within SLAs can be a challenge
  • you now have two tables: one with current but slow data another with lagging but fast data
A Vision for Analytics with No ETL

• Goals:
  • no explicit loading/conversion step to move raw data into a target table
  • a single view of the data that is
    • up-to-date
    • (mostly) in an efficient columnar format
A Vision for Analytics with No ETL

- Elements of an ETL-light analytic stack:
  - support for complex/nested schemas
    -> avoid remapping of raw data into a flat relational schema
  - background and incremental data conversion
    -> retain in-place single view of entire data set, with most data being in an efficient format
  - bonus: schema inference and schema evolution
    -> start analyzing data as soon as it arrives, regardless of its complexity
Support for Complex Schemas in Impala

- Standard relational: all columns have scalar values: CHAR(n), DECIMAL(p, s), INT, DOUBLE, TIMESTAMP, etc.
- Complex types: structs, arrays, maps in essence, a nested relational schema
- Supported file formats: Parquet, json, XML, Avro
- Design principle for SQL extensions: maintain SQL’s way of dealing with multi-valued data
Support for Complex Schemas in Impala

• Example:
CREATE TABLE Customers (  
cid BIGINT,  
address STRUCT {  
   street STRING,  
   zip INT  
},  
orders ARRAY<STRUCT {  
   oid BIGINT,  
   total DECIMAL(9, 2),  
   items ARRAY< STRUCT {  
      iid BIGINT, qty INT, price DECIMAL(9, 2) }  
}>  
)
Support for Complex Schemas in Impala

- Total revenue with items that cost more than $10:
  SELECT SUM(i.price * i.qty)
  FROM Customers.orders.items i
  WHERE i.price > 10

- Customers and order totals in zip 94611:
  SELECT c.cid, o.total
  FROM Customers c, c.orders o
  WHERE c.address.zip = 94611
Support for Complex Schemas in Impala

- Customers that have placed more than 10 orders:
  SELECT c.cid
  FROM Customers c
  WHERE COUNT(c.orders) > 10
  (shorthand for:
  WHERE (SELECT COUNT(*) FROM c.orders) > 10)

- Number of orders and average item price per customer:
  SELECT c.cid, COUNT(c.orders),
  AVG(c.orders.items.price)
  FROM Customers c
Background Format Conversion

• Sample workflow:
  • create table for data:
    CREATE TABLE LogData (…) WITH CONVERSION TO PARQUET;
  • load data into table:
    LOAD DATA INPATH ‘/raw-log-data/file1’ INTO LogData
    SOURCE FORMAT SEQUENCEFILE;

• Pre-requisite for incremental conversion:
  multi-format tables and partitions
  • currently: each table partition has a single file format
  • instead: allow a mix of file formats (separated into format-specific subdirectories)
Background Format Conversion

- Conversion process
  - atomic: the switch from the source to the target data files is atomic from the perspective of a running query (but any running query sees the full data set)
  - redundant: with option to retain original data
  - incremental: Impala’s catalog service detects new data files that are not in the target format automatically
Schema Inference and Schema Evolution

• Schema inference from data files is useful to reduce the barrier to analyzing complex source data
  • as an example, log data often has hundreds of fields
  • the time required to create the DDL manually is substantial

• Example: schema inference from structured data files
  • available today:
    CREATE TABLE LogData LIKE PARQUET '/log-data.pq'
  • future formats: XML, json, Avro
Schema Inference and Schema Evolution

• Schema evolution:
  • a necessary follow-on to schema inference: every schema evolves over time; explicit maintenance is as time-consuming as the initial creation
  • algorithmic schema evolution requires sticking to generally safe schema modifications: adding new fields
    • adding new top-level columns
    • adding fields within structs

• Example workflow:
  LOAD DATA INPATH ‘/path’ INTO LogData SOURCE FORMAT JSON WITH SCHEMA EXPANSION;
  • scans data to determine new columns/fields to add
  • synchronous: if there is an error, the ‘load’ is aborted and the user notified
Conclusion

• Hadoop offers a number of advantages over traditional multi-platform ETL solutions:
  • availability of all data sets on a single platform
  • data becomes accessible through SQL as soon as it lands
• However, this can be improved further:
  • a richer analytic SQL that is extended to handle nested data
  • an automated background conversion process that preserves an up-to-date view of all data while providing BI-typical performance
  • simple automation of initial schema creation and subsequent maintenance that makes dealing with large, complex schemas less labor-intensive
Thank you
This an example segue slide on a blue background. This could also be a quote slide.

This is an optional subtitle or space for attribution
This an example segue slide on a blue background. This could also be a quote slide.

This is an optional subtitle or space for attribution
This an example segue slide on a blue background. This could also be a quote slide.

This is an optional subtitle or space for attribution