Integration of Apache Hive and HBase

Enis Soztutar
enis[at]apache[dot]org
@enissoz
Agenda

- Overview of Hive and HBase
- Hive + HBase Features and Improvements
- Future of Hive and HBase
- Q&A
Apache Hive Overview

- Apache Hive is a data warehouse system for Hadoop
- SQL-like query language called HiveQL
- Built for PB scale data
- Main purpose is analysis and ad hoc querying
- Database / table / partition / bucket – DDL Operations
- SQL Types + Complex Types (ARRAY, MAP, etc)
- Very extensible
- Not for: small data sets, low latency queries, OLTP
Overview of Apache HBase

• Apache HBase is the Hadoop database
• Modeled after Google’s BigTable
• A sparse, distributed, persistent multi-dimensional sorted map
• The map is indexed by a row key, column key, and a timestamp
• Each value in the map is an un-interpreted array of bytes
• Low latency random data access
Overview of Apache HBase

• Logical view:

From: Bigtable: A Distributed Storage System for Structured Data, Chang, et al.
Apache HBase Architecture

Client

HMaster

Region server
Region
Region
Region server
Region
Region
Region server
Region
Region

Zookeeper

HDFS
Hive + HBase Features and Improvements
Hive + HBase Motivation

• Hive and HBase has different characteristics:
  - High latency vs. Low latency
  - Structured vs. Unstructured
  - Analysts vs. Programmers

• Hive datawarehouses on Hadoop are high latency
  - Long ETL times
  - Access to real time data

• Analyzing HBase data with MapReduce requires custom coding

• Hive and SQL are already known by many analysts
Use Case 1: HBase as ETL Data Sink

From HUG - Hive/HBase Integration or, MaybeSQL? April 2010 John Sichi Facebook
http://www.slideshare.net/hadoopusergroup/hive-h-basehadoopapr2010
Use Case 2: HBase as Data Source
Use Case 3: Low Latency Warehouse
Example: Hive + Hbase (HBase table)

```
hbase(main):001:0> create 'short_urls', {NAME => 'u'}, {NAME=>'s'}

hbase(main):014:0> scan 'short_urls'

ROW           COLUMN+CELL
bit.ly/aaaaa  column=s:hits, value=100
bit.ly/aaaaa  column=u:url,
value=hbase.apache.org/
bit.ly/abcd   column=s:hits, value=123
bit.ly/abcd   column=u:url,
value=example.com/foo
```
Example: Hive + HBase (Hive table)

CREATE TABLE short_urls(
    short_url string,
    url string,
    hit_count int
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'

WITH SERDEPROPERTIES
("hbase.columns.mapping" = ":key, u:url, s:hits")

TBLPROPERTIES
("hbase.table.name" = "short_urls");
Storage Handler

• Hive defines HiveStorageHandler class for different storage backends: HBase/ Cassandra / MongoDB/ etc

• Storage Handler has hooks for
  – Getting input / output formats
  – Meta data operations hook: CREATE TABLE, DROP TABLE, etc

• Storage Handler is a table level concept
  – Does not support Hive partitions, and buckets
Hive + HBase Integration

- For Input/OutputFormat, getSplits(), etc underlying HBase classes are used
- Column selection and certain filters can be pushed down
- HBase tables can be used with other (Hadoop native) tables and SQL constructs
- Hive DDL operations are converted to HBase DDL operations via the client hook.
  - All operations are performed by the client
  - No two phase commit
Schema / Type Mapping
Schema Mapping

- Hive table + columns + column types <=> HBase table + column families (+ column qualifiers)
- Every field in Hive table is mapped in order to either
  - The table key (using :key as selector)
  - A column family (cf:) -> MAP fields in Hive
  - A column (cf:cq)
- Hive table does not need to include all columns in HBase

```sql
CREATE TABLE short_urls(
    short_url string,
    url string,
    hit_count int,
    props, map<string,string> )
WITH SERDEPROPERTIES  
("hbase.columns.mapping" = ":key, u:url, s:hits, p:")
```
Type Mapping

• Recently added to Hive (0.9.0)
• Previously all types were being converted to strings in HBase
• Hive has:
  – Primitive types: INT, STRING, BINARY, DATE, etc
  – ARRAY<Type>
  – MAP<PrimitiveType, Type>
  – STRUCT<a:INT, b:STRING, c:STRING>
• HBase does not have types
  – Bytes.toBytes()
Type Mapping

- Table level property
  "hbase.table.default.storage.type" = "binary"
- Type mapping can be given per column after #
  - Any prefix of "binary", eg u:url#b
  - Any prefix of "string", eg u:url#s
  - The dash char "-", eg u:url#-

CREATE TABLE short_urls(
  short_url string,
  url string,
  hit_count int,
  props, map<string,string>
) WITH SERDEPROPERTIES
  ("hbase.columns.mapping" = ":key#b,u:url#b,s:hits#b,p:#s")
Type Mapping

• If the type is not a primitive or Map, it is converted to a JSON string and serialized

• Still a few rough edges for schema and type mapping:
  – No Hive BINARY support in HBase mapping
  – No mapping of HBase timestamp (can only provide put timestamp)
  – No arbitrary mapping of Structs / Arrays into HBase schema
Bulk Load

• Steps to bulk load:
  – Sample source data for range partitioning
  – Save sampling results to a file
  – Run CLUSTER BY query using HiveHFileOutputFormat and TotalOrderPartitioner
  – Import Hfiles into HBase table

• Ideal setup should be

  SET hive.hbase.bulk=true

  INSERT OVERWRITE TABLE web_table SELECT ....
Filter Pushdown

• Idea is to pass down filter expressions to the storage layer to minimize scanned data
• To access indexes at HDFS or HBase
• Example:

```sql
CREATE EXTERNAL TABLE users (userid LONG, email STRING, ... ) 
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = "\:key,...")

SELECT ... FROM users WHERE userid > 1000000 and email LIKE '%@gmail.com';
```

-> `scan.setStartRow(Bytes.toBytes(1000000))`
Filter Decomposition

- Optimizer pushes down the predicates to the query plan
- Storage handlers can negotiate with the Hive optimizer to decompose the filter

\[ x > 3 \text{ AND } \text{upper}(y) = 'XYZ' \]

- Handle \( x > 3 \), send \( \text{upper}(y) = 'XYZ' \) as residual for Hive
- Works with:
  
  \[ \text{key} = 3, \text{key} > 3, \text{etc} \]

  \[ \text{key} > 3 \text{ AND } \text{key} < 100 \]

- Only works against constant expressions
Security – Big Picture

• Security becomes more important to support enterprise level and multi tenant applications
• 5 Different Components to ensure / impose security
  - HDFS
  - MapReduce
  - HBase
  - Zookeeper
  - Hive
• Each component has:
  - Authentication
  - Authorization
HBase Security – Closer look

- Released with HBase 0.92
- Fully optional module, disabled by default
- Needs an underlying secure Hadoop release
- SecureRPCEngine: optional engine enforcing SASL authentication
  - Kerberos
  - DIGEST-MD5 based tokens
  - TokenProvider coprocessor
- Access control is implemented as a Coprocessor: AccessController
- Stores and distributes ACL data via Zookeeper
  - Sensitive data is only accessible by HBase daemons
  - Client does not need to authenticate to zk
Hive Security – Closer look

- Hive has different deployment options, security considerations should take into account different deployments
- Authentication is only supported at Metastore, not on HiveServer, web interface, JDBC
- Authorization is enforced at the query layer (Driver)
- Pluggable authorization providers. Default one stores global/table/partition/column permissions in Metastore

```
GRANT ALTER ON TABLE web_table TO USER bob;
CREATE ROLE db_reader
GRANT SELECT, SHOW_DATABASE ON DATABASE mydb TO ROLE db_reader
```
Hive Deployment Option 1
Hive Deployment Option 2
Hive + HBase + Hadoop Security

• Regardless of Hive’s own security, for Hive to work on secure Hadoop and HBase, we should:
  – Obtain delegation tokens for Hadoop and HBase jobs
  – Ensure to obey the storage level (HDFS, HBase) permission checks
  – In HiveServer deployments, authenticate and impersonate the user

• Delegation tokens for Hadoop are already working

• Obtaining HBase delegation tokens are released in Hive 0.9.0
Future of Hive + HBase

- Improve on schema / type mapping
- Fully secure Hive deployment options
- HBase bulk import improvements
- Sortable signed numeric types in HBase
- Filter pushdown: non key column filters
- Hive random access support for HBase
References

• **Security**
  - https://issues.apache.org/jira/browse/HIVE-2764
  - https://issues.apache.org/jira/browse/HBASE-5371
  - https://issues.apache.org/jira/browse/HCATALOG-245
  - https://issues.apache.org/jira/browse/HCATALOG-260
  - https://issues.apache.org/jira/browse/HCATALOG-244
  - https://cwiki.apache.org/confluence/display/HCATALOG/Hcat+Security+Design

• **Type mapping / Filter Pushdown**
  - https://issues.apache.org/jira/browse/HIVE-1634
  - https://issues.apache.org/jira/browse/HIVE-1226
  - https://issues.apache.org/jira/browse/HIVE-1643
  - https://issues.apache.org/jira/browse/HIVE-2815
  - https://issues.apache.org/jira/browse/HIVE-1643
Other Resources

• Hadoop Summit
  – June 13-14
  – San Jose, California

• Hadoop Training and Certification
  – Developing Solutions Using Apache Hadoop
  – Administering Apache Hadoop
  – Online classes available US, India, EMEA
Thanks

Questions?