Complex stories about Sqooping PostgreSQL data

10/28/2013
NTT DATA Corporation
Masatake Iwasaki
Introduction
About Me

Masatake Iwasaki:
   Software Engineer @ NTT DATA:
         NTT(Nippon Telegraph and Telephone Corporation): Telecommunication
         NTT DATA: Systems Integrator

Developed:
   Ludia: Fulltext search index for PostgreSQL using Senna

Authored:
   “A Complete Primer for Hadoop”
   (no official English title)

Patches for Sqoop:
   SQOOP-390: PostgreSQL connector for direct export with pg_bulkload
   SQOOP-999: Support bulk load from HDFS to PostgreSQL using COPY ... FROM
   SQOOP-1155: Sqoop 2 documentation for connector development
Why PostgreSQL?

Enterprisy
from earlier version
comparing to MySQL

Active community in Japan

NTT DATA commits itself to development
Sqooping PostgreSQL data
Direct connector for PostgreSQL loader:
SQOOP-390: PostgreSQL connector for direct export with pg_bulkload

Yet another direct connector for PostgreSQL JDBC:
SQOOP-999: Support bulk load from HDFS to PostgreSQL using COPY ... FROM

Supporting complex data types:
SQOOP-1149: Support Custom Postgres Types
Direct connector for PostgreSQL loader

SQOOP-390:
PostgreSQL connector for direct export with pg_bulkload

pg_bulkload:
Data loader for PostgreSQL
Server side plug-in library and client side command
Providing filtering and transformation of data
http://pgbulkload.projects.pgfoundry.org/
SQOOP-390: PostgreSQL connector for direct export with pg_bulkload

**Diagram:**
- HDFS
  - File Split
  - Mapper
    - pg_bulkload
    - tmp1
  - Mapper
    - pg_bulkload
    - tmp2
  - Mapper
    - pg_bulkload
    - tmp3

- PostgreSQL
  - tmp1
  - tmp2
  - tmp3

- Reducer
  - Destination Table

- CREATE TABLE tmp3(LIKE dest INCLUDING CONSTRAINTS)
- staging table per mapper is must due to table level locks
- BEGIN
  - INSERT INTO dest ( SELECT * FROM tmp1 )
  - DROP TABLE tmp1
  - INSERT INTO dest ( SELECT * FROM tmp2 )
  - DROP TABLE tmp2
  - INSERT INTO dest ( SELECT * FROM tmp3 )
  - DROP TABLE tmp3
  - COMMIT

- external process
Direct connector for PostgreSQL loader

Pros:
- Fast
  - by short-circuitting server functionality
- Flexible
  - filtering error records

Cons:
- Not so fast
  - Bottleneck is not in client side but in DB side
  - Built-in COPY functionality is fast enough
- Not General
  - pg_bulkload supports only export
- Requiring setup on all slave nodes and client node
- Possible to Require recovery on failure
Yet another direct connector for PostgreSQL JDBC

PostgreSQL provides custom SQL command for data import/export

```
COPY table_name [ ( column_name [, ...] ) ]
  FROM { 'filename' | STDIN }
  [ [ WITH ] ( option [, ...] ) ]

COPY { table_name [ ( column_name [, ...] ) ] | ( query ) }
  TO { 'filename' | STDOUT }
  [ [ WITH ] ( option [, ...] ) ]
```

where option can be one of:

- FORMAT format_name
- OIDS [ boolean ]
- DELIMITER 'delimiter_character'
- NULL 'null_string'
- HEADER [ boolean ]
- QUOTE 'quote_character'
- ESCAPE 'escape_character'
- FORCE_QUOTE { ( column_name [, ...] ) | * }
- FORCE_NOT_NULL ( column_name [, ...] )
- ENCODING 'encoding_name'

AND JDBC API

```
org.postgresql.copy.*
```
SQOOP-999: Support bulk load from HDFS to PostgreSQL using COPY ... FROM

Using custom SQL command via JDBC API only available in PostgreSQL

COPY FROM STDIN WITH ...
import org.postgresql.copy.CopyManager;
import org.postgresql.copy.CopyIn;
...

protected void setup(Context context)
...
    dbConf = new DBConfiguration(conf);
    CopyManager cm = null;

...

public void map(LongWritable key, Writable value, Context context)
...
    if (value instanceof Text) {
        line.append(System.getProperty("line.separator"));
    }
    try {
        byte[] data = line.toString().getBytes("UTF-8");
        copyin.writeToCopy(data, 0, data.length);
    }
Yet another direct connector for PostgreSQL JDBC

Pros:
- Fast enough
- Ease of use
  - JDBC driver jar is distributed automatically by MR framework

Cons:
- Dependency on not general JDBC
  - possible licensing issue (PostgreSQL is OK, it’s BSD Lisence)
  - build time requirement (PostgreSQL JDBC is available in Maven repo.)
  
  <dependency org="org.postgresql" name="postgresql"
  rev="${postgresql.version}" conf="common->default" />

- Error record causes rollback of whole transaction
- Still difficult to implement custom connector for IMPORT because of code generation part
Supporting complex data types:

PostgreSQL supports lot of complex data types

Geometric Types
- Points
- Line Segments
- Boxes
- Paths
- Polygons
- Circles

Network Address Types
- inet
- cidr
- macaddr

XML Type

JSON Type

Supporting complex data types:
SQOOP-1149: Support Custom Postgres Types
Constraints on JDBC data types in Sqoop framework

protected Map<String, Integer> getColumnTypesForRawQuery(String stmt) {
    ... 
    results = execute(stmt); 
    ... 
    ResultSetMetaData metadata = results.getMetaData(); 
    for (int i = 1; i < cols + 1; i++) {
        int typeId = metadata.getColumnType(i);
    }

    public String toJavaType(int sqlType) {
        // Mappings taken from:
        if (sqlType == Types.INTEGER) {
            return "Integer";
        } else if (sqlType == Types.VARCHAR) {
            return "String";
        } else {
            // TODO(aaron): Support DISTINCT, ARRAY, STRUCT, REF, JAVA_OBJECT.
            // Return null indicating database-specific manager should return a java data type if it can find one for any nonstandard type.
            return null;
        }
    }

    returns java.sql.Types.OTHER for types not mappable to basic Java data types => Losing type information

    reaches here
Sqoop 1 Summary

Pros:

Simple Standalone MapReduce Driver
Easy to understand for MR application developers

Variety of connectors
Lot of information

Cons:

Complex command line and inconsistent options
meaning of options is according to connectors
Not enough modular
Dependency on JDBC data model
Security
Sqoop 2

Everything are rewritten
Working on server side
More modular

Not compatible with Sqoop 1 at all
(Almost) Only generic connector
Black box comparing to Sqoop 1
Needs more documentation

SQOOP-1155: Sqoop 2 documentation for connector development

Internal of Sqoop2 MapReduce Job

- OutputFormat invokes Loader's load method (via SqoopOutputFor...

.. todo: sequence diagram like figure.
Sqoop2: Initialization phase of IMPORT job
Sqoop2: Map phase of IMPORT job

run

read from DB

extract

write*

context.write

Converter to Sqoop internal data format

Implement this

SqoopMapper

Extractor

MapDataWriter
Sqoop2: Reduce phase of EXPORT job

Implement this

Reducer\[SqoopNullOutputFormat\]

\[getRecordWriter\]

\[run\]

\[write\]

\[setContent\]

\[getRecordWriter\]

\[getter\]

\[load\]

\[read\]

\[write into DB\]
My interest for popularizing Sqoop 2

Complex data type support in Sqoop 2
Bridge to use Sqoop 1 connectors on Sqoop 2
Bridge to use Sqoop 2 connectors from Sqoop 1 CLI