What is HIVE?

- A data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.
  - ETL.
  - Structure.
  - Access to different storage.
  - Query execution via MapReduce.

- Key Building Principles:
  - SQL is a familiar language
  - Extensibility – Types, Functions, Formats, Scripts
  - Performance

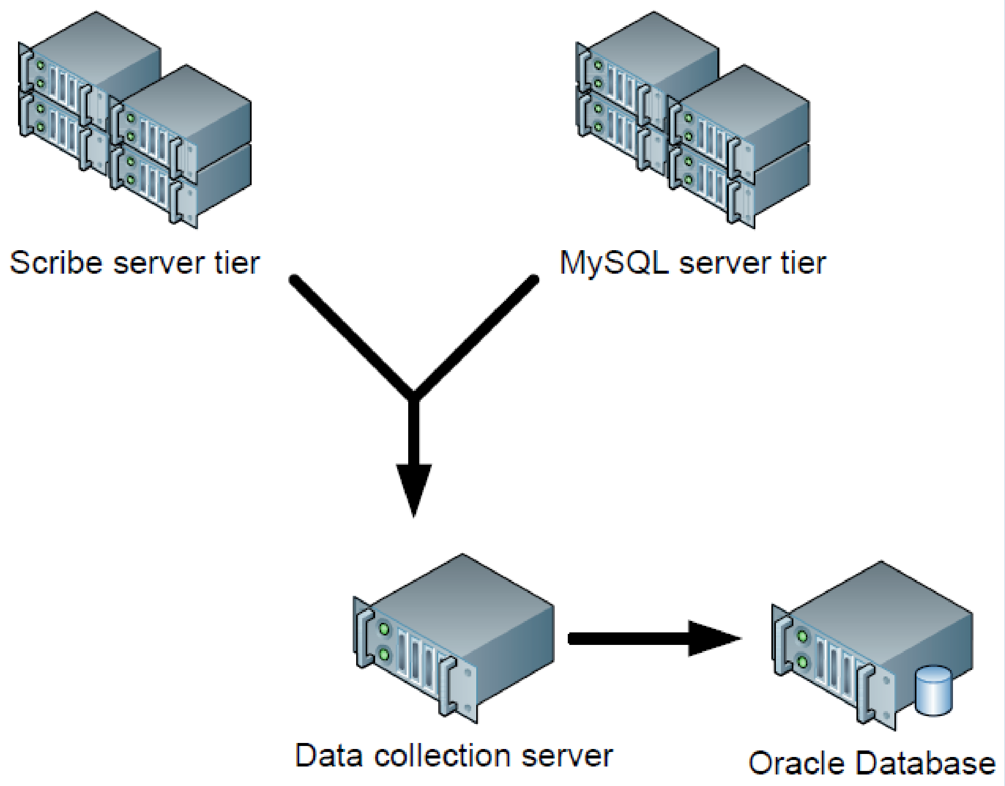
Overview

- Intuitive
  - Make the unstructured data looks like tables regardless how it really lay out
  - SQL based query can be directly against these tables
  - Generate specify execution plan for this query

- What's Hive
  - A data warehousing system to store structured data on Hadoop file system
  - Provide an easy query these data by execution Hadoop MapReduce plans

Background

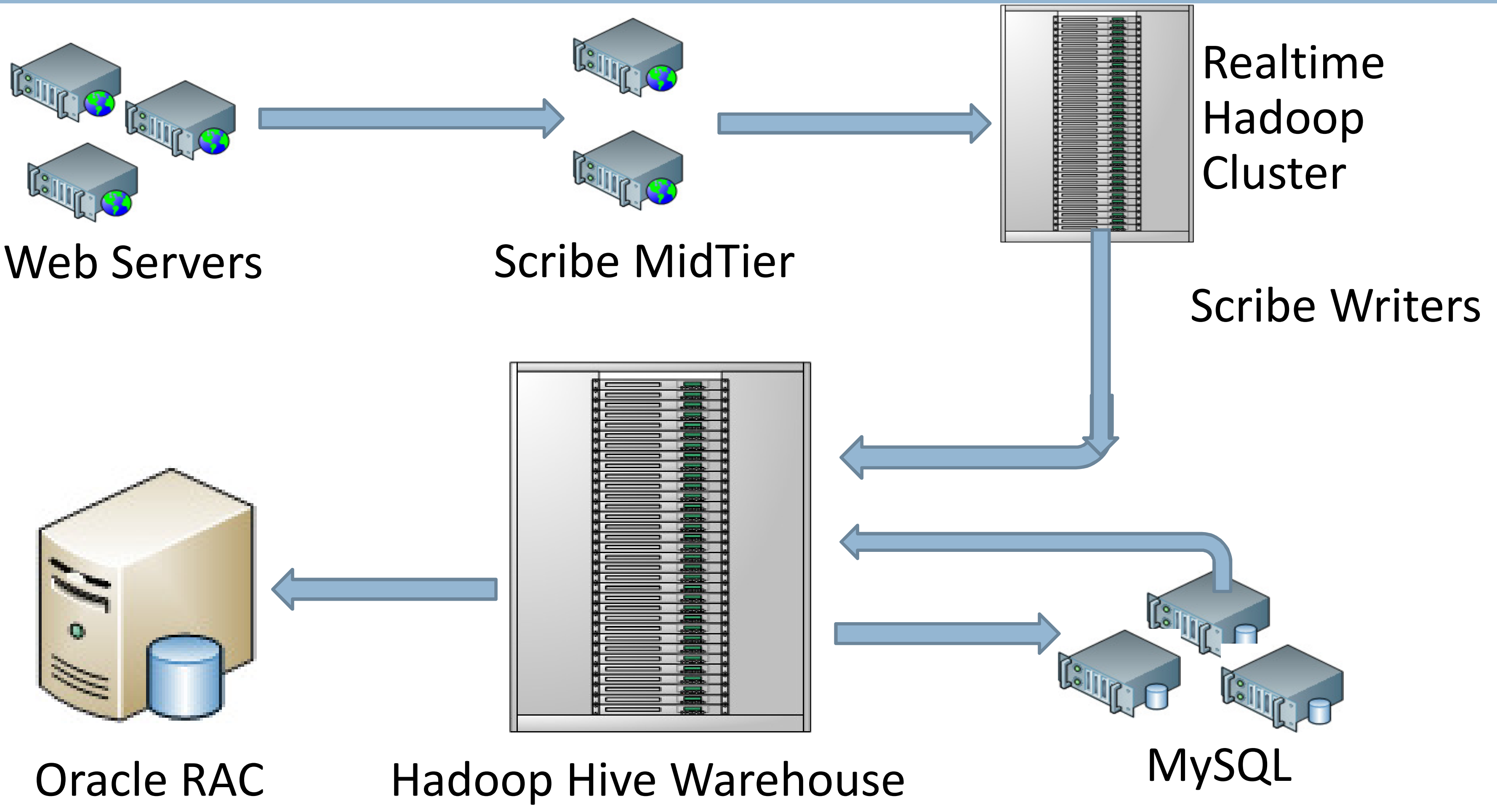
- Started at Facebook
- Data was collected by nightly cron jobs into Oracle DB
- "ETL" via hand-coded python
- Grew from 10s of GBs (2006) to 1 TB/day new data (2007), now 10x that.



Hive Applications

- Log processing
- Text mining
- Document indexing
- Customer-facing business intelligence (e.g., Google Analytics)
- Predictive modeling, hypothesis testing

Motivation

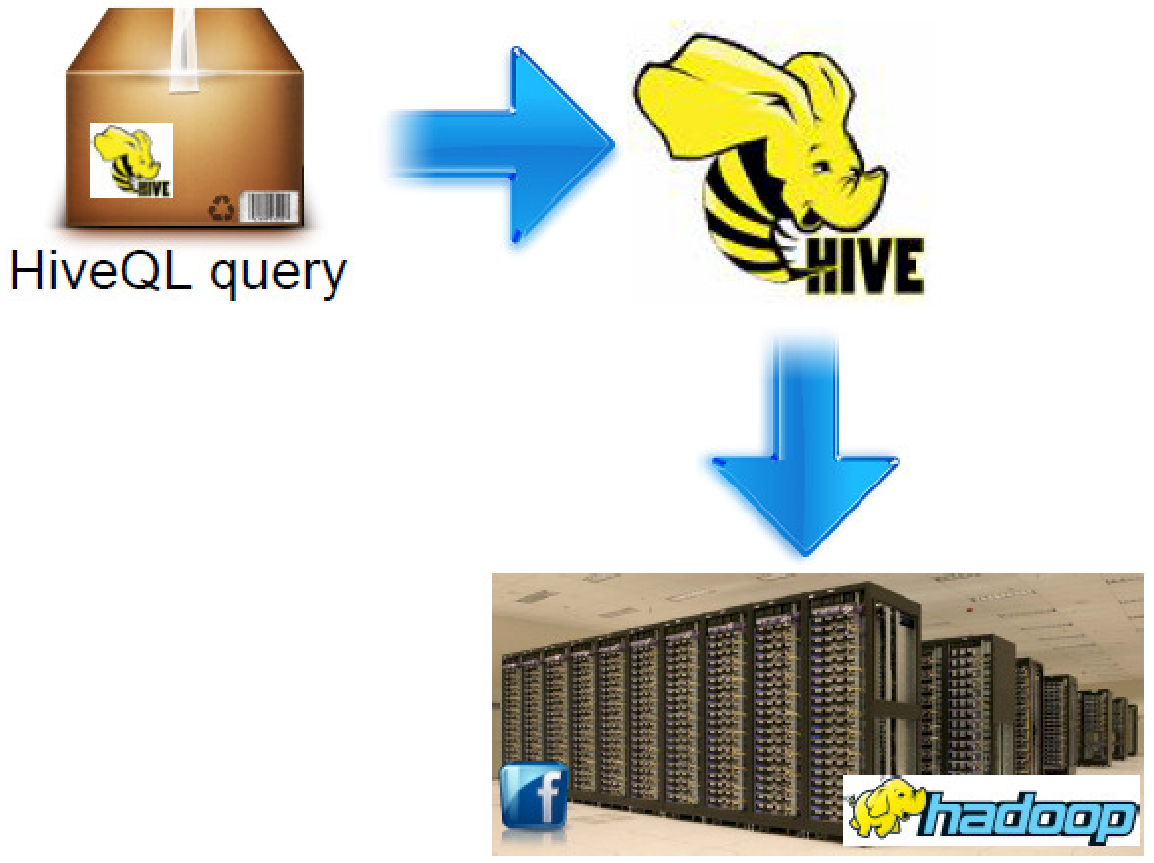

Web Servers
Scribe MidTier
Realtime Hadoop Cluster
Scribe Writers
Oracle RAC
Hadoop Hive Warehouse
MySQL

Motivation

- Analysis of Data made by both engineering and non-engineering people.
- The data are growing fast.
- Current RDBMS can NOT handle it.
- Current solution are not available, not scalable, Expensive and Proprietary.
- Hadoop supports data-intensive distributed applications.
- However...
  - Map-reduce hard to program (users know sql/bash/python).
  - No schema.

How does Apache Hive works?

- Hive is built on top of Hadoop
- Hive stores data in the HDFS
- Hive compile SQL queries as MapReduce jobs and run the jobs in the cluster

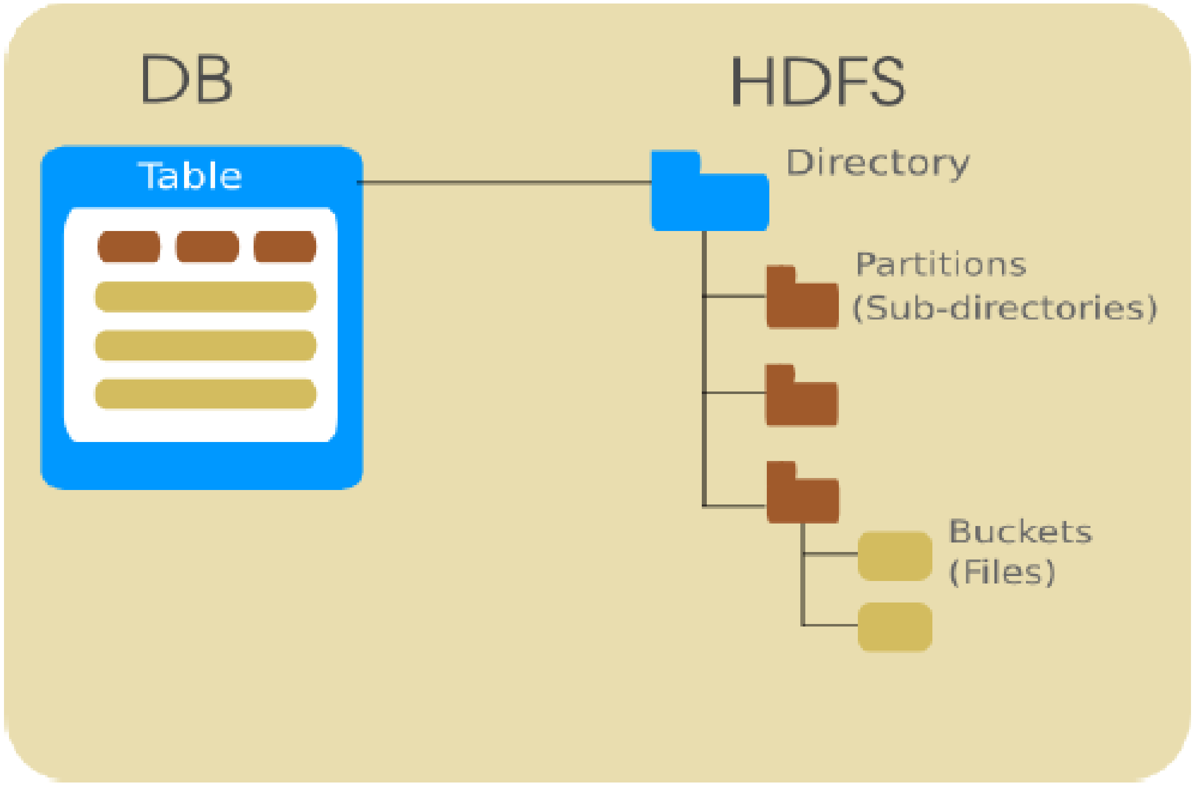


Motivation

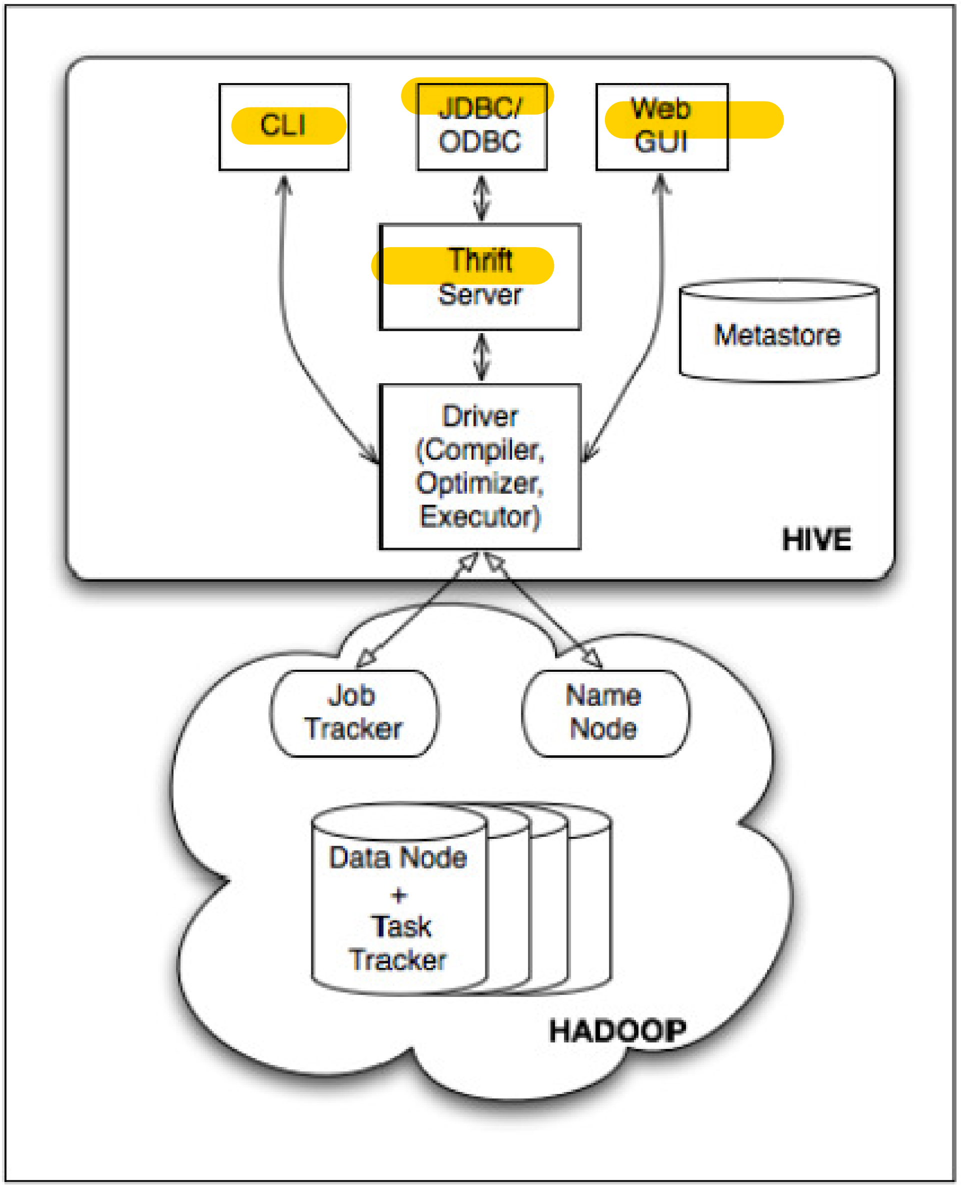
- Limitation of MR
  - Have to use M/R model
  - Not Reusable
  - Error prone
  - For complex jobs:
    - Multiple stage of Map/Reduce functions
    - Just like ask dev to write specify physical execution plan in the database

Data Model

- Hive structures data into database concepts like tables, columns, rows.
- Tables
  - Basic type columns (int, float, boolean)
  - Complex type: List / Map (associate array)
- Partitions
- Buckets



Architecture



Architecture

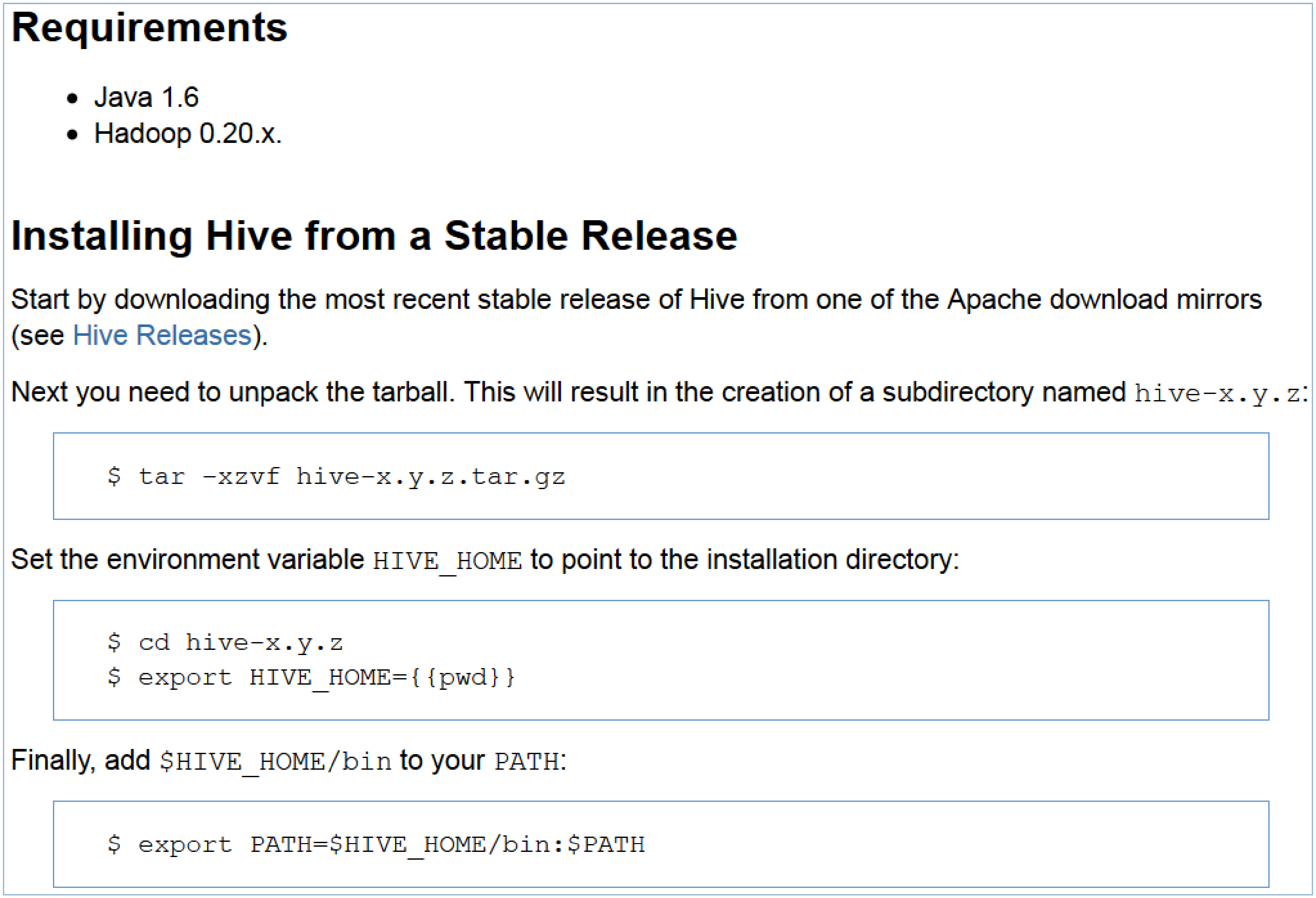
- **External Interfaces**
  - provides both user interfaces like command line (CLI) and web UI, and application programming interfaces (API) like JDBC and ODBC
- **Thrift Server**
  - exposes a very simple client API to execute HiveQL statements
- **Metastore**
  - is the system catalog.
  - All other components of Hive interact with the metastore.
- **Driver**
  - manages the life cycle of a HiveQL statement during compilation, optimization and execution
- **Compiler**
  - translates statements into a plan which consists of a DAG of map-reduce jobs
- The driver submits the individual map-reduce jobs from the DAG to the **Execution Engine** in a topological order

Metastore

- The metastore is the system catalog which contains metadata about the tables stored in Hive.
- **Database**
  - is a namespace for tables.
- **Table**
  - Metadata for table contains list of columns and their types, owner, storage and SerDe information
- **Partition**
  - Each partition can have its own columns and SerDe and storage information

Query Compiler

- **Parser**
  - transforms a query string to a parse tree representation.
- **Semantic Analyzer**
  - transforms the parse tree to a blockbased internal query representation.
- **Logical Plan Generator**
  - converts the internal query representation to a logical plan, which consists of a tree of logical operators
- **Optimizer**
  - performs multiple passes over the logical plan and rewrites it in several ways
- **Physical Plan Generator**
  - converts the logical plan into a physical plan, consisting of a DAG of map-reduce jobs



Requirements

- Java 1.6
- Hadoop 0.20.x.

Installing Hive from a Stable Release

Start by downloading the most recent stable release of Hive from one of the Apache download mirrors (see Hive Releases).

Next you need to unpack the tarball. This will result in the creation of a subdirectory named `hive-x.y.z`:

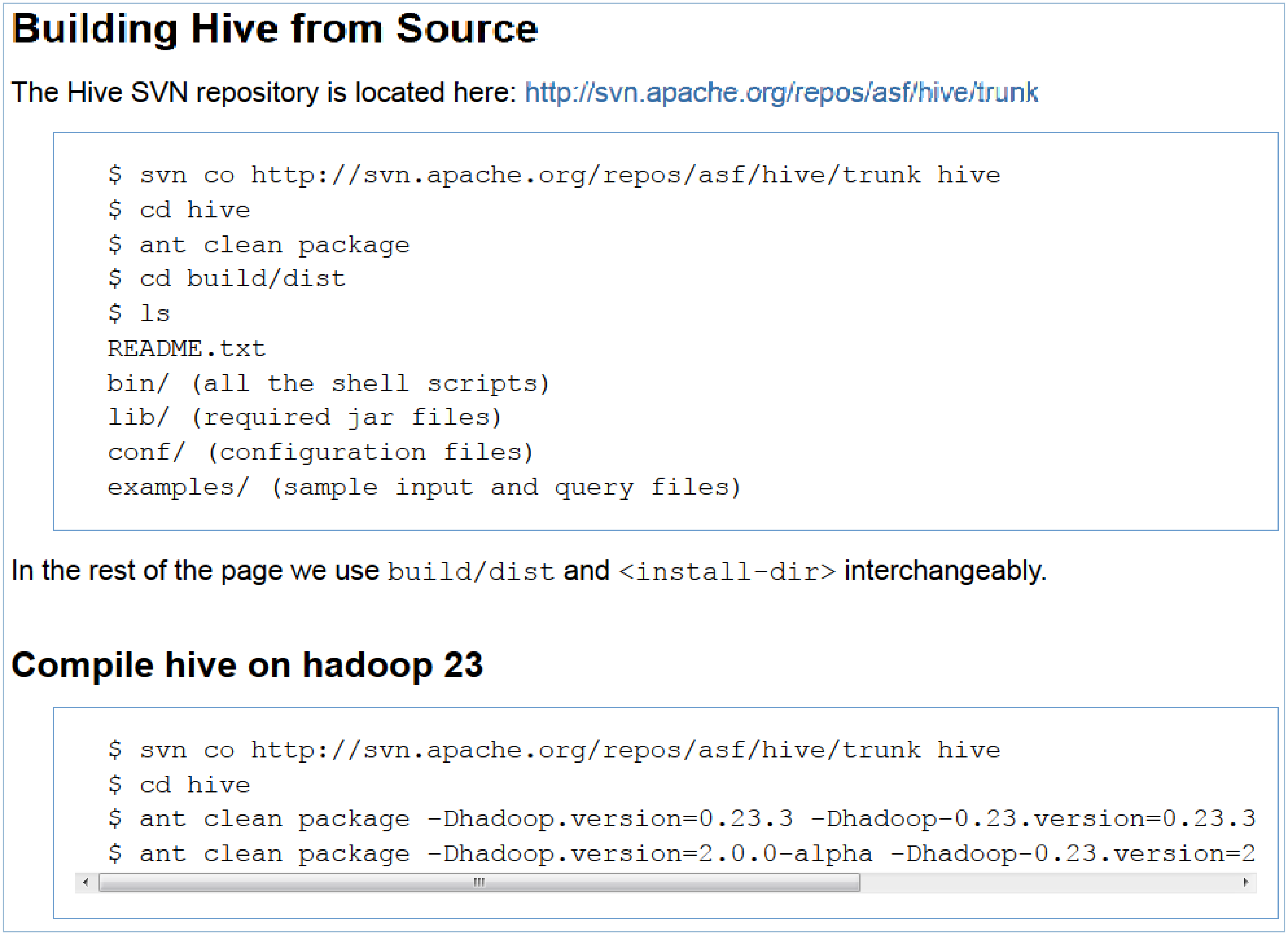
```
$ tar -xzvf hive-x.y.z.tar.gz
```

Set the environment variable `HIVE_HOME` to point to the installation directory:

```
$ cd hive-x.y.z
$ export HIVE_HOME={{pwd}}
```

Finally, add `$HIVE_HOME/bin` to your `PATH`:

```
$ export PATH=$HIVE_HOME/bin:$PATH
```



Building Hive from Source

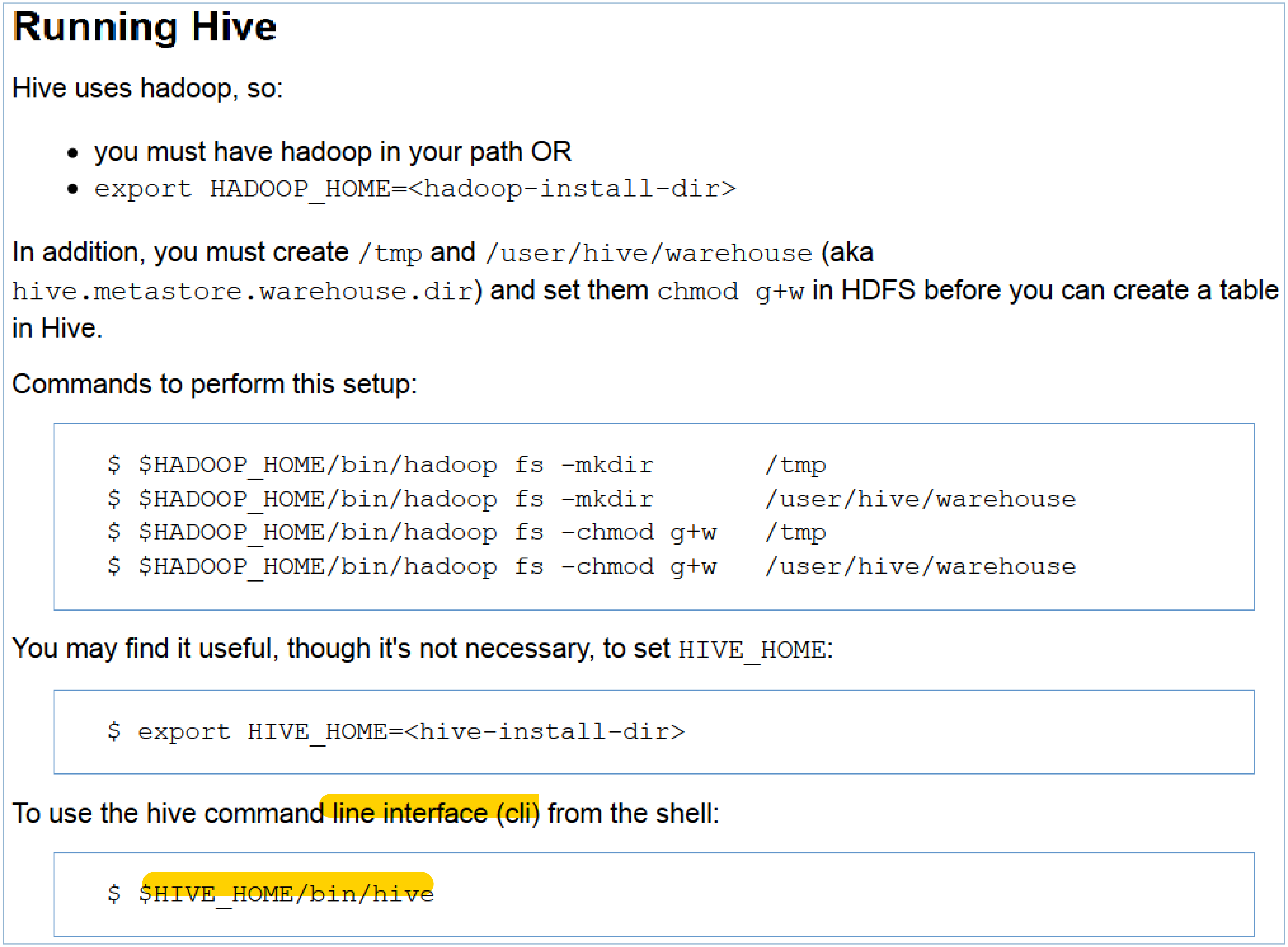
The Hive SVN repository is located here: http://svn.apache.org/repos/asf/hive/trunk

```
$ svn co http://svn.apache.org/repos/asf/hive/trunk hive
$ cd hive
$ ant clean package
$ cd build/dist
$ ls
README.txt
bin/ (all the shell scripts)
lib/ (required jar files)
conf/ (configuration files)
examples/ (sample input and query files)
```

In the rest of the page we use `build/dist` and `<install-dir>` interchangeably.

Compile hive on hadoop 23

```
$ svn co http://svn.apache.org/repos/asf/hive/trunk hive
$ cd hive
$ ant clean package -Dhadoop.version=0.23.3 -Dhadoop-0.23.version=0.23.3
$ ant clean package -Dhadoop.version=2.0.0-alpha -Dhadoop-0.23.version=2
```



Running Hive

Hive uses hadoop, so:

- you must have hadoop in your path OR
- `export HADOOP_HOME=<hadoop-install-dir>`

In addition, you must create `/tmp` and `/user/hive/warehouse` (aka `hive.metastore.warehouse.dir`) and set them `chmod g+w` in HDFS before you can create a table in Hive.

Commands to perform this setup:

```
  $ $HADOOP_HOME/bin/hadoop fs -mkdir       /tmp
  $ $HADOOP_HOME/bin/hadoop fs -mkdir       /user/hive/warehouse
  $ $HADOOP_HOME/bin/hadoop fs -chmod g+w   /tmp
  $ $HADOOP_HOME/bin/hadoop fs -chmod g+w   /user/hive/warehouse
```

You may find it useful, though it's not necessary, to set `HIVE_HOME`:

```
  $ export HIVE_HOME=<hive-install-dir>
```

To use the hive command line interface (cli) from the shell:

```
  $ $HIVE_HOME/bin/hive
```

Configuration management overview

- Hive by default gets its configuration from `<install-dir>/conf/hive-default.xml`
- The location of the Hive configuration directory can be changed by setting the `HIVE_CONF_DIR` environment variable.
- Configuration variables can be changed by (re-)defining them in `<install-dir>/conf/hive-site.xml`
- Log4j configuration is stored in `<install-dir>/conf/hive-log4j.properties`

- Hive configuration is an overlay on top of hadoop – it inherits the hadoop configuration variables by default.

- Hive configuration can be manipulated by:
  - Editing hive-site.xml and defining any desired variables (including hadoop variables) in it
  - From the cli using the set command (see below)
  - Invoking hive using the syntax:
    - `$ bin/hive -hiveconf x1=y1 -hiveconf x2=y2`
      this sets the variables x1 and x2 to y1 and y2 respectively
  - Setting the `HIVE_OPTS` environment variable to "-hiveconf x1=y1 -hiveconf x2=y2" which does the same as above.

Runtime configuration

- Hive queries are executed using map-reduce queries and, therefore, the behavior of such queries can be controlled by the hadoop configuration variables.
- The cli command 'SET' can be used to set any hadoop (or hive) configuration variable. For example:

```
hive> SET mapred.job.tracker=myhost.mycompany.com:50030;
hive> SET -v;
```

The latter shows all the current settings. Without the `-v` option only the variables that differ from the base hadoop configuration are displayed.

DDL Operations

The Hive DDL operations are documented in Hive Data Definition Language.

Creating Hive Tables

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
```

creates a table called pokes with two columns, the first being an integer and the other a string.

```
hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);
```

creates a table called invites with two columns and a partition column called ds. The partition column is a virtual column. It is not part of the data itself but is derived from the partition that a particular dataset is loaded into.

By default, tables are assumed to be of text input format and the delimiters are assumed to be ^A(ctrl-a).

Browsing through Tables

```
hive> SHOW TABLES;
```

lists all the tables.

```
hive> SHOW TABLES '.*s';
```

lists all the table that end with 's'. The pattern matching follows Java regular expressions. Check out this link for documentation http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html.

```
hive> DESCRIBE invites;
```

shows the list of columns.

Altering and Dropping Tables

Table names can be changed and columns can be added or replaced:

```
hive> ALTER TABLE events RENAME TO 3koobecaf;
hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
hive> ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT 'a comment');
hive> ALTER TABLE invites REPLACE COLUMNS (foo INT, bar STRING, baz INT COMMENT 'baz repl
```

Note that REPLACE COLUMNS replaces all existing columns and only changes the table's schema, not the data. The table must use a native SerDe. REPLACE COLUMNS can also be used to drop columns from the table's schema:

```
hive> ALTER TABLE invites REPLACE COLUMNS (foo INT COMMENT 'only keep the first column');
```

Dropping tables:

```
hive> DROP TABLE pokes;
```

DML Operations

The Hive DML operations are documented in Hive Data Manipulation Language.

Loading data from flat files into Hive:

```
hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;
```

Loads a file that contains two columns separated by ctrl-a into pokes table. 'LOCAL' signifies that the input file is on the local file system. If 'LOCAL' is omitted then it looks for the file in HDFS.

The keyword 'OVERWRITE' signifies that existing data in the table is deleted. If the 'OVERWRITE' keyword is omitted, data files are appended to existing data sets.

NOTES:

- NO verification of data against the schema is performed by the load command.
- If the file is in hdfs, it is moved into the Hive-controlled file system namespace.
  The root of the Hive directory is specified by the option `hive.metastore.warehouse.dir` in `hive-default.xml`. We advise users to create this directory before trying to create tables via Hive.

```
hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');
hive> LOAD DATA LOCAL INPATH './examples/files/kv3.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-08');
```

The two LOAD statements above load data into two different partitions of the table invites. Table invites must be created as partitioned by the key ds for this to succeed.

```
hive> LOAD DATA INPATH '/user/myname/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');
```

The above command will load data from an HDFS file/directory to the table.
Note that loading data from HDFS will result in moving the file/directory. As a result, the operation is almost instantaneous.

SQL Operations

The Hive query operations are documented in Select.

Example Queries

Some example queries are shown below. They are available in `build/dist/examples/queries`.
More are available in the hive sources at `ql/src/test/queries/positive`.

SELECTS and FILTERS

```
hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';
```

selects column 'foo' from all rows of partition `ds=2008-08-15` of the `invites` table. The results are not stored anywhere, but are displayed on the console.

Note that in all the examples that follow, `INSERT` (into a hive table, local directory or HDFS directory) is optional.

```
hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a WHERE a.ds='2008-08-15';
```

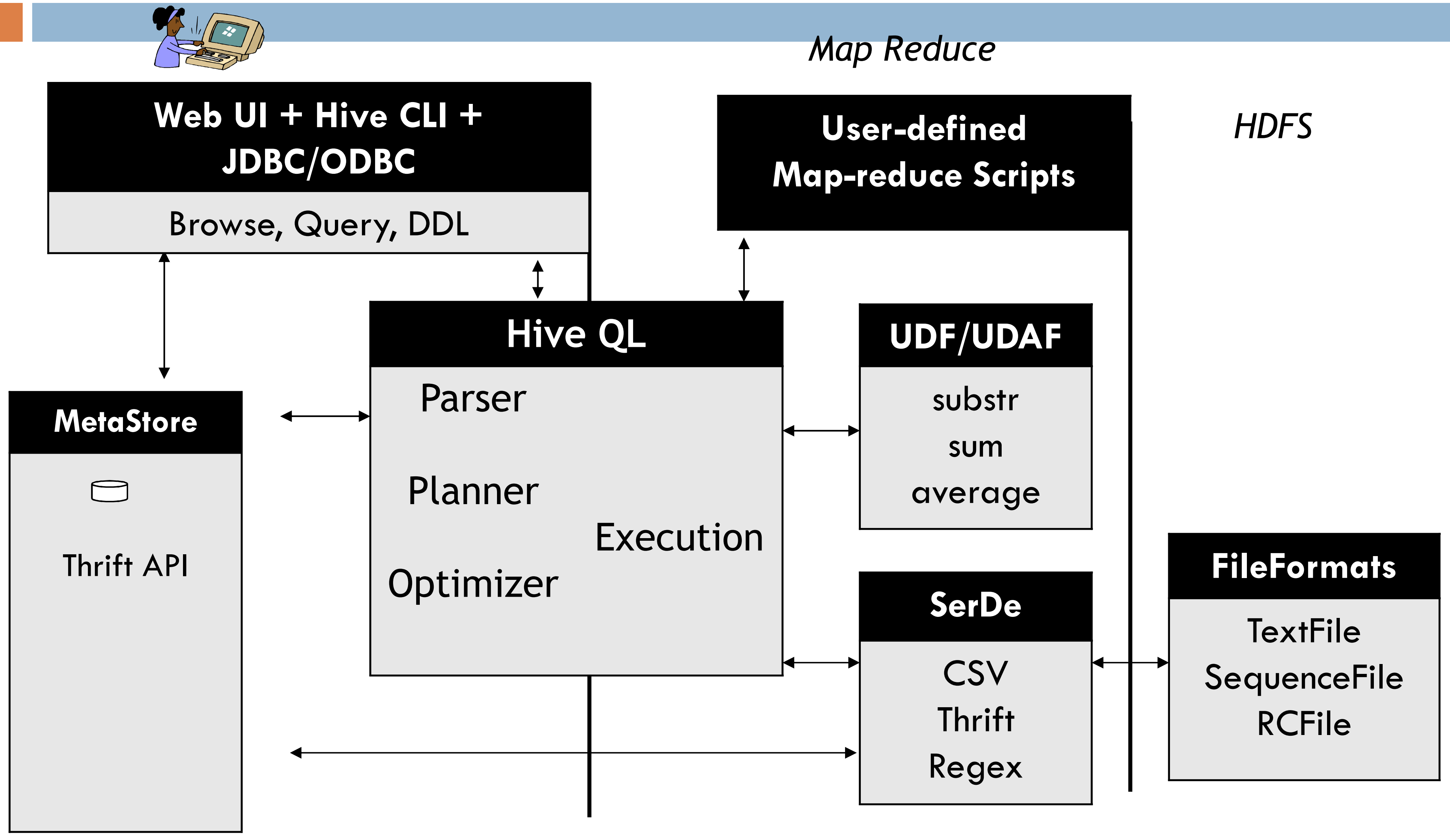
selects all rows from partition `ds=2008-08-15` of the `invites` table into an HDFS directory. The result data is in files (depending on the number of mappers) in that directory.
NOTE: partition columns if any are selected by the use of *. They can also be specified in the projection clauses.

Partitioned tables must always have a partition selected in the `WHERE` clause of the statement.

```
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out' SELECT a.* FROM pokes a;
```

selects all rows from pokes table into a local directory.

Architecture



Performance

- GROUP BY operation
  - Efficient execution plans based on:
    - Data skew:
      - how evenly distributed data across a number of physical nodes
      - bottleneck VS load balance
    - Partial aggregation:
      - Group the data with the same group by value as soon as possible
      - In memory hash-table for mapper
      - Earlier than combiner

Performance

- JOIN operation
  - Traditional Map-Reduce Join
  - Early Map-side Join
    - very efficient for joining a small table with a large table
    - Keep smaller table data in memory first
    - Join with a chunk of larger table data each time
    - Space complexity for time complexity

Performance

- Ser/De
  - Describe how to load the data from the file into a representation that make it looks like a table;
- Lazy load
  - Create the field object when necessary
  - Reduce the overhead to create unnecessary objects in Hive
  - Java is expensive to create objects
  - Increase performance

Hive – Performance

Date	SVN Revision	Major Changes	Query A	Query B	Query C
2/22/2009	746906	Before Lazy Deserialization	83 sec	98 sec	183 sec
2/23/2009	747293	Lazy Deserialization	40 sec	66 sec	185 sec
3/6/2009	751166	Map-side Aggregation	22 sec	67 sec	182 sec
4/29/2009	770074	Object Reuse	21 sec	49 sec	130 sec
6/3/2009	781633	Map-side Join *	21 sec	48 sec	132 sec
8/5/2009	801497	Lazy Binary Format *	21 sec	48 sec	132 sec

- QueryA: SELECT count(1) FROM t;
- QueryB: SELECT concat(concat(concat(a,b),c),d) FROM t;
- QueryC: SELECT * FROM t;
- map-side time only (incl. GzipCodec for comp/decompression)
- * These two features need to be tested with other queries.

Pros

- Pros
  - A easy way to process large scale data
  - Support SQL-based queries
  - Provide more user defined interfaces to extend
  - Programmability
  - Efficient execution plans for performance
  - Interoperability with other database tools

Cons

- Cons
  - No easy way to append data
  - Files in HDFS are immutable
- Future work
  - Views / Variables
  - More operator
    - In/Exists semantic
  - More future work in the mail list

Application

- Log processing
  - Daily Report
  - User Activity Measurement
- Data/Text mining
  - Machine learning (Training Data)
- Business intelligence
  - Advertising Delivery
  - Spam Detection

Related Work

- Parallel databases: Gamma, Bubba, Volcano
- Google: Sawzall
- Yahoo: Pig
- IBM: JAQL
- Microsoft: DradLINQ , SCOPE

Hive Components

- Shell Interface: Like the MySQL shell
- Driver:
  - Session handles, fetch, exeucition
- Complier:
  - Prarse,plan,optimzie
- Execution Engine:
  - DAG stage
  - Run map or reduce

Motivation

- MapReduce Motivation
  - Data processing: > 1 TB
  - Massively parallel
  - Locality
  - Fault Tolerant

Hive Usage

- hive> show tables;
- hive> create table SHAKESPEARE (freq INT,word STRING) row format delimited fields terminated by ‘\t’ stored as textfile
- hive> load data inpath “shakespeare_freq” into table shakespeare;

Hive Usage

- hive> load data inpath “shakespeare_freq” into table shakespeare;
- hive> select * from shakespeare where freq>100 sort by freq asc limit 10;

Hive Usage @ Facebook

- Statistics per day:
  - 4 TB of compressed new data added per day
  - 135TB of compressed data scanned per day
  - 7500+ Hive jobs on per day

- Hive simplifies Hadoop:
  - ~200 people/month run jobs on Hadoop/Hive
  - Analysts (non-engineers) use Hadoop through Hive
  - 95% of jobs are Hive Jobs

Data Units

- Databases.
- Tables.
- Partitions.
- Buckets (or Clusters).

Type System

- Primitive types
  - Integers:TINYINT, SMALLINT, INT, BIGINT.
  - Boolean: BOOLEAN.
  - Floating point numbers: FLOAT, DOUBLE .
  - String: STRING.

- Complex types
  - Structs: {a INT; b INT}.
  - Maps: M['group'].
  - Arrays: ['a', 'b', 'c'], A[1] returns 'b'.

Examples – DDL Operations

- CREATE TABLE sample (foo INT, bar STRING) PARTITIONED BY (ds STRING);
- SHOW TABLES '.*s';
- DESCRIBE sample;
- ALTER TABLE sample ADD COLUMNS (new_col INT);
- DROP TABLE sample;

Examples – DML Operations

- LOAD DATA LOCAL INPATH './sample.txt' OVERWRITE INTO TABLE sample PARTITION (ds='2012-02-24');

- LOAD DATA INPATH '/user/falvariz/hive/sample.txt' OVERWRITE INTO TABLE sample PARTITION (ds='2012-02-24');

SELECTS and FILTERS

- SELECT foo FROM sample WHERE ds='2012-02-24';
- INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT * FROM sample WHERE ds='2012-02-24';
- INSERT OVERWRITE LOCAL DIRECTORY '/tmp/hive-sample-out' SELECT * FROM sample;

Aggregations and Groups

- SELECT MAX(foo) FROM sample;
- SELECT ds, COUNT(*), SUM(foo) FROM sample GROUP BY ds;
- FROM sample s INSERT OVERWRITE TABLE bar SELECT s.bar, count(*) WHERE s.foo > 0 GROUP BY s.bar;

Join

```
CREATE TABLE customer (id INT,name STRING,address STRING)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '#';

CREATE TABLE order_cust (id INT,cus_id INT,prod_id INT,price INT)
 ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

- SELECT * FROM customer c JOIN order_cust o ON (c.id=o.cus_id);
- SELECT c.id, c.name, c.address, ce.exp FROM customer c JOIN (SELECT cus_id,sum(price) AS exp FROM order_cust GROUP BY cus_id) ce ON (c.id=ce.cus_id);

Multi table insert – Dynamic partition insert

```
FROM page_view_stg pvs
   INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08', country='US')
          SELECT pvs.viewTime, … WHERE pvs.country = 'US'
   INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08', country='CA')
          SELECT pvs.viewTime, ... WHERE pvs.country = 'CA'
   INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08', country='UK')
          SELECT pvs.viewTime, ... WHERE pvs.country = 'UK';

FROM page_view_stg pvs
   INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08', country)
          SELECT pvs.viewTime, ...
```

User-defined function

- **Java code**

```
package com.example.hive.udf;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public final class Lower extends UDF {
  public Text evaluate(final Text s) {
    if (s == null) { return null; }
    return new Text(s.toString().toLowerCase());
  }
}
```

- **Registering the class**
  - **CREATE FUNCTION** my_lower **AS** 'com.example.hive.udf.Lower';

- **Using the function**
  - **SELECT** my_lower(title), sum(freq) **FROM** titles **GROUP BY** my_lower(title);

Built-in Functions

- Mathematical:
  - round, floor, ceil, rand, exp...
- Collection:
  - size, map_keys, map_values, array_contains.
- Type Conversion:
  - cast.
- Date:
  - from_unixtime, to_date, year, datediff...
- Conditional:
  - if, case, coalesce.
- String:
  - length, reverse, upper, trim...

More Functions

- Aggregate: count,sum,variance...
- Table-Generating:
  - Lateral view:

string pageid	Array<int> adid_list
"front_page"	[1,2,3]
"contact_page"	[3, 4, 5]

SELECT pageid, adid
FROM pageAds **LATERAL VIEW** explode(adid_list) adTable **AS** adid;

string pageid	int adid
"front_page"	1
"front_page"	2
...	...

Map/Reduce Scripts

- my_append.py

```
for line in sys.stdin:
  line = line.strip()
  key = line.split('\t')[0]
  value = line.split('\t')[1]
  print key+str(i)+'\t'+value+str(i)
  i=i+1
```

- Using the function

  SELECT TRANSFORM (foo, bar) USING 'python ./my_append.py' FROM sample;

Comparison of UDF/UDAF v.s. M/R scripts

	UDF/UDAF	M/R scripts
language	Java	any language
1/1 input/output	supported via UDF	supported
n/1 input/output	supported via UDAF	supported
1/n input/output	supported via UDTF	supported
Speed	Faster (in same process)	Slower (spawns new process)

Performance - Dataset structure

grep(key VARCHAR(10), field VARCHAR(90))	2 columns, 500 million rows, 50GB
rankings(pageRank INT, pageURL VARCHAR(100), avgDuration INT)	3 columns, 56.3 million rows, 3.3GB.
uservisits(sourceIP VARCHAR(16), destURL VARCHAR(100), visitDate DATE, adRevenue FLOAT, userAgent VARCHAR(64), countryCode VARCHAR(3), languageCode VARCHAR(6), searchWord VARCHAR(32), duration INT).	9 columns, 465 million rows, 60GB (scaled down from 200GB).

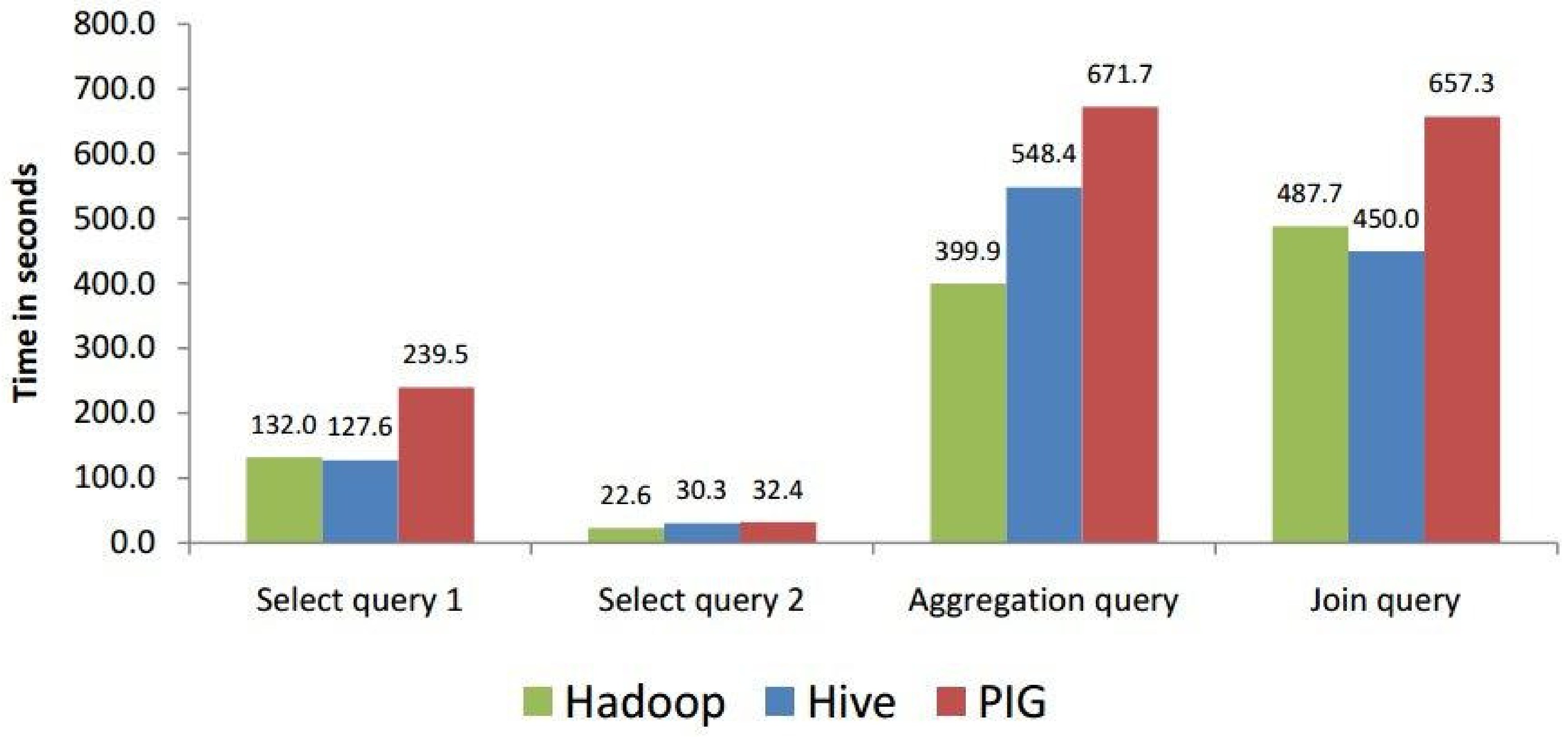
Performance - Test query

Select query 1	SELECT * FROM grep WHERE field like ‘%XYZ%’;
Select query 2	SELECT pageRank, pageURL FROM rankings WHERE pageRank > 10;

Aggregation query	SELECT sourceIP, SUM(adRevenue)
	FROM uservisits GROUP BY sourceIP;

Join query	SELECT INTO Temp sourceIP,
	AVG(pageRank) as avgPageRank,
	SUM(adRevenue) as totalRevenue
	FROM rankings AS R, userVisits AS UV
	WHERE R.pageURL = UV.destURL
	AND UV.visitDate BETWEEN Date(\`1999-01-01') AND Date(\`2000-01-01')
	GROUP BY UV.sourceIP;

Performance - Result



Conclusion

- A easy way to process large scale data.
  - Hive provides a solution to perform business intelligence of huge data on top of mature Hadoop map-reduce platform.
- Support SQL-based queries.
  - The SQL-like HiveQL cuts off the learning curve compared with low-level map-reduce programs.
- Provide more user defined interfaces to extend
- Programmability.
- Files in HDFS are immutable. Typically:
  - Log processing: Daily Report, User Activity Measurement
  - Data/Text mining: Machine learning (Training Data)
  - Business intelligence: Advertising Delivery,Spam Detection