# Content

- Big Data Analytics and Data Science
- Map-reduce and Hadoop fundamentals
- Map-reduce and Hadoop pros and cons
- Hadoop Success Stories
- Hadoop Evolution and Hadoop eco-system
- Pig and Hive
- Map Reduce Cloud based solutions
- Where is the world going? Spark Apache project

# Need for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Not everyone wants to (or can) write Java code

- Solution: develop higher-level data processing languages
  - Pig: Pig Latin is a bit like Perl
  - Hive: HQL is like SQL

# Pig and Hive

- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source


- Common idea:
  - Higher-level language “compiles down” to Hadoop jobs

# Big data analysis workflow

**Data Collection**

**Data Factory**
**Pig**

Pipelines
Iterative Processing

**Data Warehouse**
**Hive**

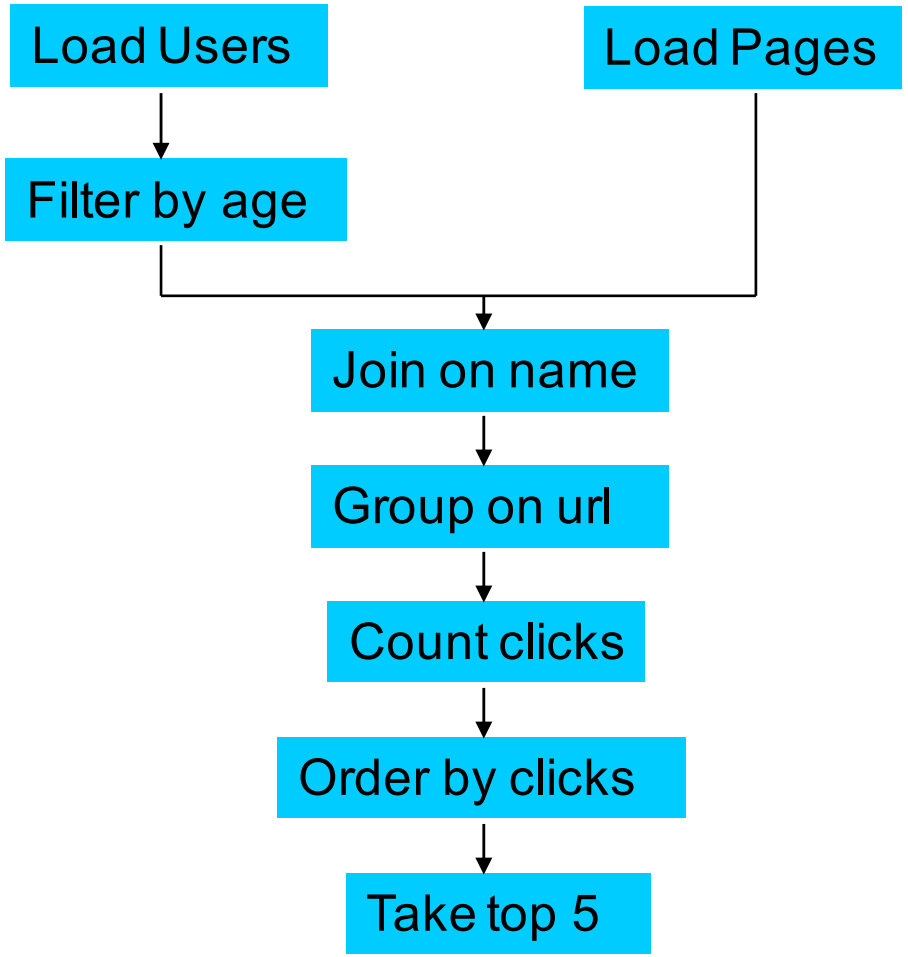
BI Tools
Analysis

# High level languages: Pig

- Pig's language (**Pig Latin**) which has the following key properties:
  - **Ease of programming:** It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated **data transformations** are explicitly encoded as **dataflow sequences**, making them easy to write, understand, and maintain
  - **Optimization opportunities:** The way in which tasks are encoded permits the system to **optimize** their **execution** automatically, allowing the user to focus on semantics rather than efficiency
  - **Extensibility:** Users can **create** their **own functions** to do special-purpose processing

# Motivation by example

Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited pages by users aged 18 - 25.



# In Map Reduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see
            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top  100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

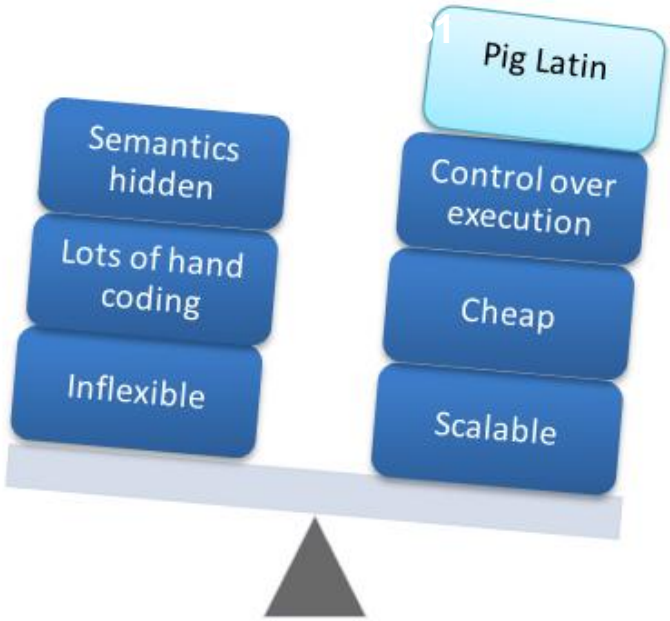
# In Pig Latin

```
Users = load 'users' as (name, age);
Fltrd = filter Users by
        age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
       COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites';
```

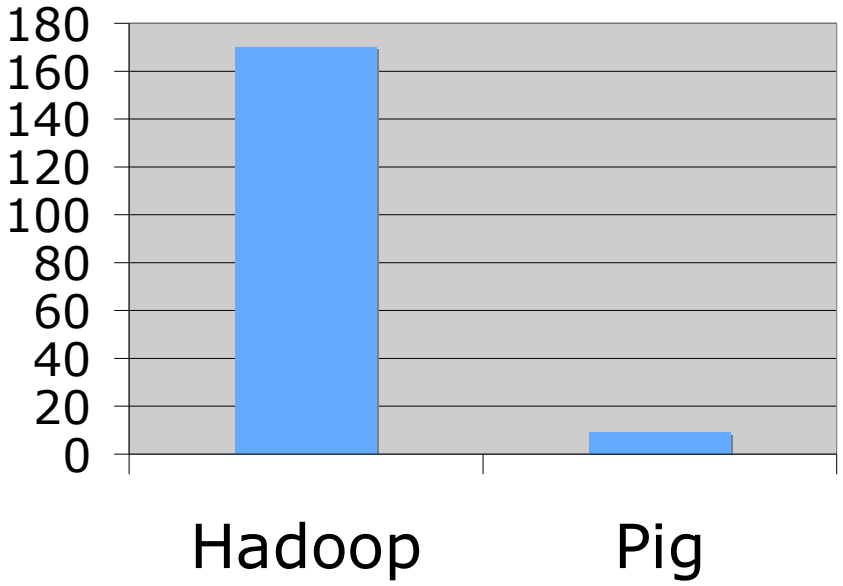
# Word Count using Pig

```
Lines=LOAD 'input/hadoop.log' AS (line:
chararray);
Words = FOREACH Lines GENERATE
FLATTEN(TOKENIZE(line)) AS word;
Groups = GROUP Words BY word;
Counts = FOREACH Groups GENERATE  group,
COUNT(Words);
Results = ORDER Words BY Counts DESC;
Top5 = LIMIT Results 5;
STORE Top5 INTO '/output/top5words';
```

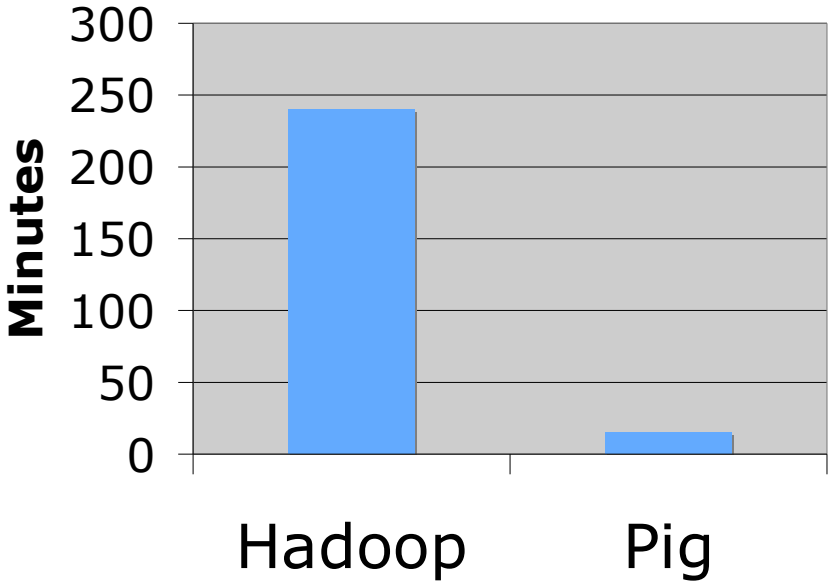
# Java vs. Pig Latin




**1/20 the lines of code**




**1/16 the development time**



# Pig Highlights

- User defined functions (UDFs) can be written for column transformation (TOUPPER), or aggregation (SUM)
- UDFs can be written to take advantage of the combiner
- Four join implementations built in: hash, fragment-replicate, merge, skewed
- Multi-query: Pig will combine certain types of operations together in a single pipeline to reduce the number of times data is scanned
- Piggybank, a collection of user contributed UDFs

# Who uses Pig for What?

- Since 2007 >50% of production jobs at Yahoo
- Also used by Twitter, LinkedIn, Ebay, AOL, …
- Used to
  - Process web logs
  - Build user behavior models
  - Process images
  - Build maps of the web
  - Do research on raw data sets

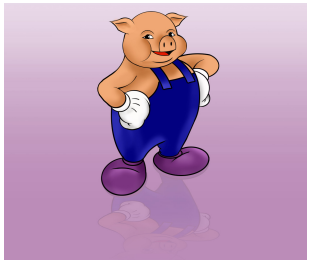
# Who uses Pig for What?

- Twitter: processing logs, mining tweet data
- AOL and MapQuest: for analytics and batch data processing
- LinkedIn: discover people you might know
- Ebay: search optimization
- Mendeley: finding trending keyword over time

# How It Works

Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
    x, COUNT(B);
STORE D INTO 'output';
```



pig.jar:

- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

Execution Plan
Map:
Filter
Count

Combine/Reduce:
Sum

# High level languages: Hive


- A **data warehouse** system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the **analysis** of large **datasets** stored in Hadoop compatible file systems
- Make the **unstructured data** looks **like tables** regardless how it really lay out
- Provides a mechanism to project structure onto this data and query the data using a **SQL-like language** called **HiveQL**
- At the same time this language also allows traditional map/reduce programmers to plug in their **custom mappers** and **reducers** when it is inconvenient or inefficient to express this logic in HiveQL

# Hive Applications @Facebook

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

# HiveQL

- It doesn't fully conform to any particular revision of the ANSI SQL standard (close to MySQL's dialect)
- No support for row-level inserts, updates, and deletes and transactions
- Still, much of HiveQL will be familiar

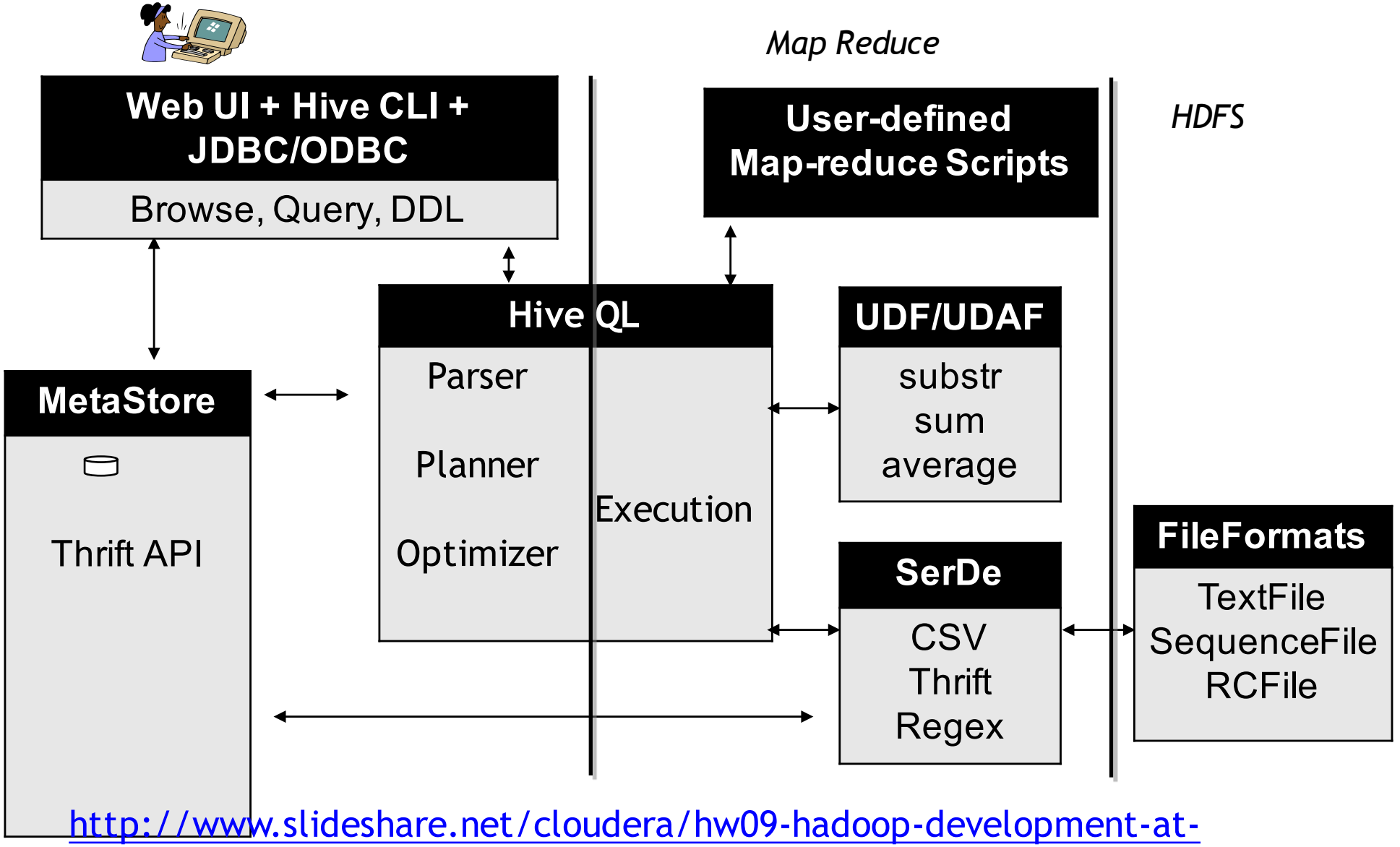
# HiveQL

- Basic SQL
  - From clause subquery
  - ANSI JOIN (equi-join only)
  - Sampling
  - Objects traversal
- Extensibility
  - Pluggable Map-reduce scripts using TRANSFORM
  - UDFs and UDAFs

# Data Model

- **Databases**: Namespaces separating tables and other data units from naming confliction
- **Tables**: Homogeneous units of data which have the same schema
- **Partitions**: Each Table can have one or more partition Keys which determines how the data is stored. Partitions also allow the user to efficiently identify the rows that satisfy a certain criteria
- **Buckets** (or **Clusters**): Data in each partition may in turn be divided into Buckets based on the value of a hash function of some column of the Table

# Architecture




http://www.slideshare.net/cloudera/hw09-hadoop-development-at-facebook-hive-and-hdfs

# MetaStore

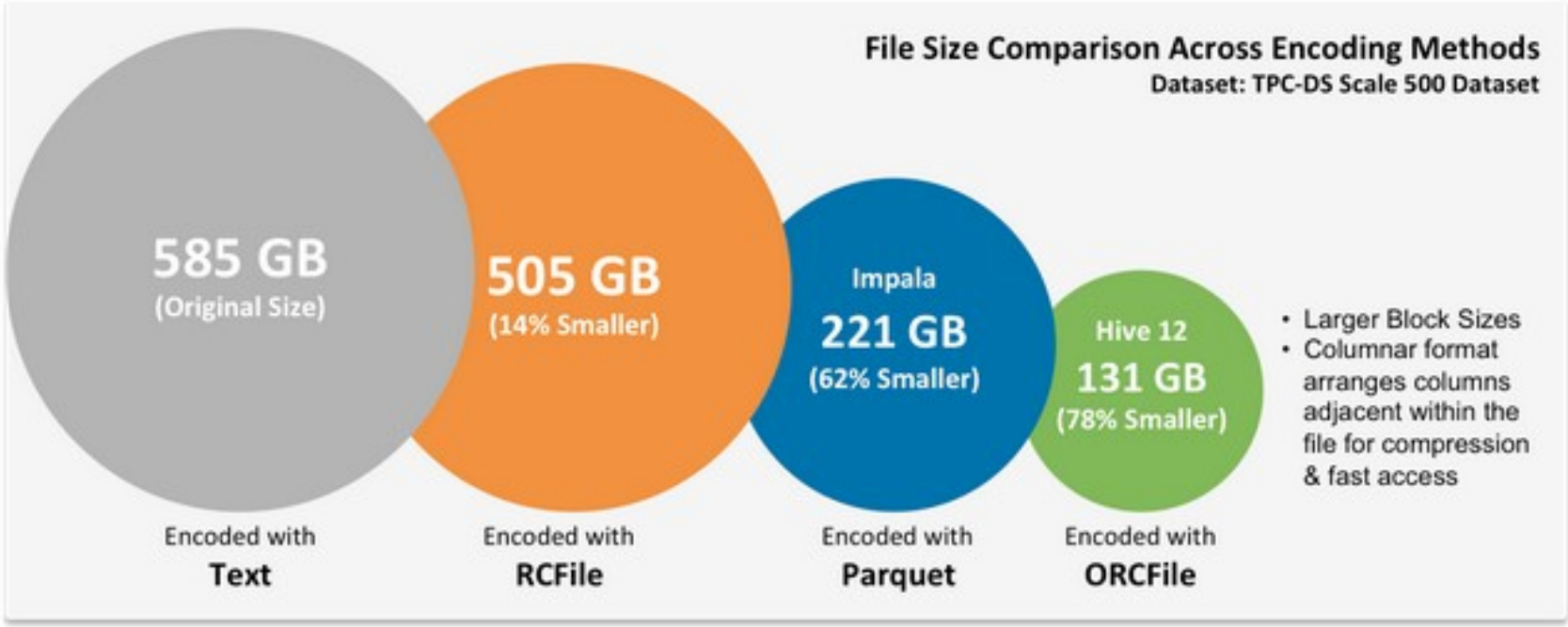
- Stores Table/Partition properties:
  - Table schema and SerDe library
  - Table Location on HDFS
  - Logical Partitioning keys and types
  - Partition level metadata
  - Other information
- Thrift API
  - Current clients in Php (Web Interface), Python interface to Hive, Java (Query Engine and CLI)
- Metadata stored in any SQL backend

# Existing File Format

| | TEXTFILE | SEQUENCEFILE | RCFILE |
|---|---|---|---|
| Data type | text only | text/binary | text/binary |
| Internal Storage order | Row-based | Row-based | Column-based |
| Compression | File-based | Block-based | Block-based |
| Splitable* | YES | YES | YES |
| Splitable* after compression | NO | YES | YES |

*** Splitable: Capable of splitting the file so that a single huge file can be processed by multiple mappers in parallel.**

# Existing File Format



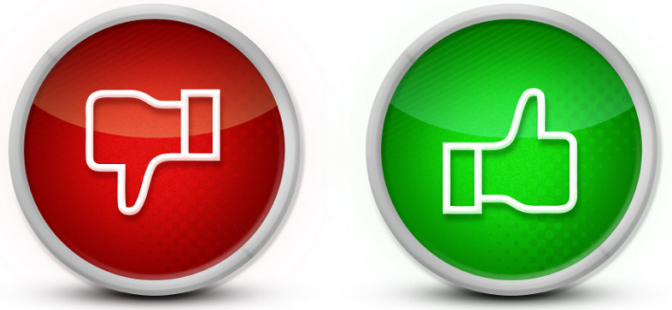

http://www.enterprisetech.com/2014/04/11/facebook-compresses-300-pb-data-warehouse/

**How Facebook Compresses Its 300 PB Data Warehouse**

April 11, 2014 by Timothy Prickett Morgan


When you have a 300 PB data warehouse running atop Hadoop, you do everything in your power to keep from adding another rack of disk drives to this beast. While social network giant Facebook is not afraid to throw a lot of hardware at a scalability problem, its software engineers are always looking for

# Pros & Cons



- Pros:
  - A easy way to process large scale data
  - Support SQL-based queries
  - Programmability
  - Efficient execution plans for performance
  - Interoperability with other database tools

- Cons:
  - No easy way to append data
  - Files in HDFS are immutable

# Pig vs. Hive

- PigLatin is procedural, where Hive is declarative
- Pig Latin allows pipeline developers to decide where to check point data in the pipeline
- PigLatin allows the developer to select specific operator implementations directly rather than relying on the optimizer
- PigLatin supports splits in the pipeline
- Pig Latin allows developers to insert their own code almost anywhere in the data pipeline
- HIVE has a MetaStore, Pig doesn’t

# Pig vs. Hive

- Use Hive when you have warehousing needs and you are good at SQL and don't want to write MapReduce jobs. But each and every problem cannot be solved using HiveQL

- Use Pig when you want to do a lot of transformations on your data and don't want to take the pain of writing MapReduce jobs