CIS 560 – Database System Concepts

Lecture 32

NoSQL – MapReduce

November 22, 2013

Credits for slides: Wisdom, Alberton, Pokorný, Hoekstra.

Reminders

- Project - DB implementation and queries due tonight
- Quiz from NoSQL lectures – 12/06

Where we are

- Today:
  - MapReduce joins and other operators
  - Hive, Pig Latin
    - Hive – A Petabyte Scale Data Warehouse Using Hadoop
    - Pig Latin: A Not-So-Foreign Language for Data Processing
- Next:
  - Key-value stores
    - Dynamo: Amazon's Highly Available Key-value Store (2007)

MapReduce Example (modified #2)

Each record: UserID, URL, timestamp, additional-info

Separate records: UserID, name, age, gender, …

Task: Total "value" of accesses for each domain based on user attributes

Joins in MapReduce

Employees

Name	Age	Dept_Id
Alex	26	2
Ben	24	2
Sara	34	5

Department

Dept_Id	Name
5	Mkt
2	Eng
3	Sales

- SELECT Employees.Name, Employees.Age, Department.Name FROM Employees INNER JOIN Department ON Employees.Dept_Id=Department.Dept_Id

http://www.javacodegeeks.com/2012/05/joins-with-map-reduce.html

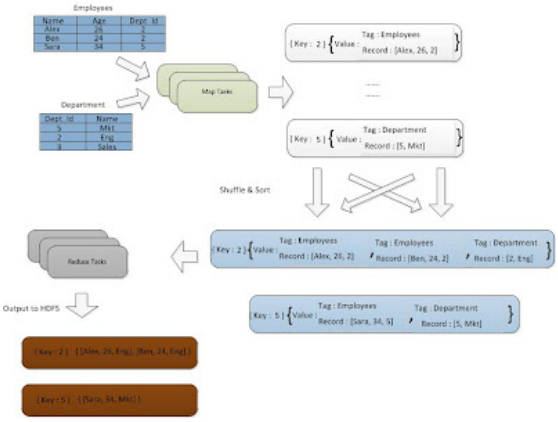
Map Side Join

- *Using Distributed Cache on Smaller Table*
  - Smaller table has to fit into memory
  - Store into a hash-table, so look-up by Dept_Id can be done easily
  - Replicate to each node and load to the memory

```
map (K table, V rec) {
list recs = lookup(rec.Dept_Id) // Get smaller table records having this Dept_Id
for (small_table_rec : recs) {
joined_rec = join (small_table_rec, rec)
}
emit (rec.Dept_id, joined_rec)
}
```

http://www.javacodegeeks.com/2012/05/joins-with-map-reduce.html

Reduce Side Join



http://www.javacodegeeks.com/2012/05/joins-with-map-reduce.html

Reduce Side Join

```
map (K table, V rec) {
    dept_id = rec.Dept_Id
    tagged_rec.tag = table
    tagged_rec.rec = rec
    emit(dept_id, tagged_rec)
}
```

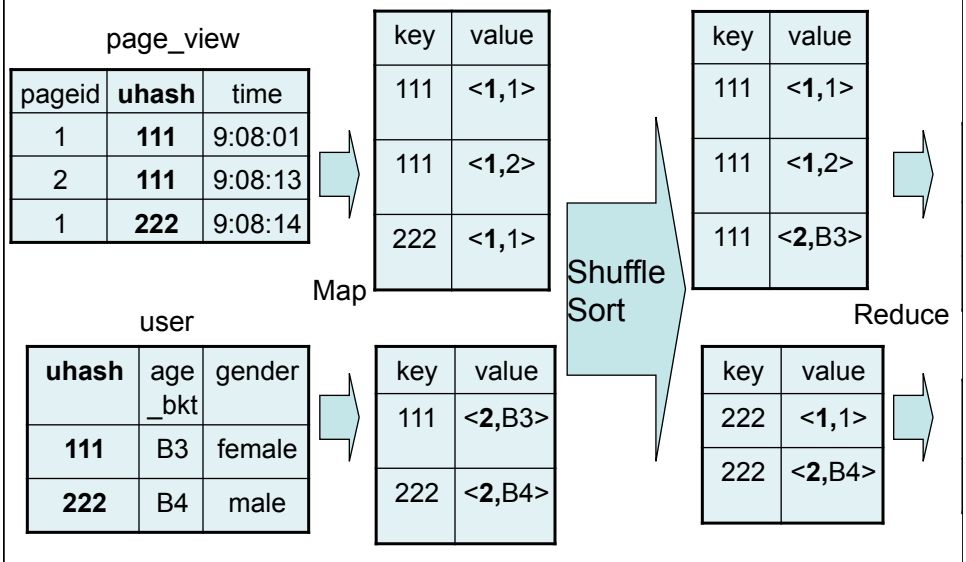
```
reduce (K dept_id, list<tagged_rec> tagged_recs) {
   for (tagged_rec : tagged_recs) {
      for (tagged_rec1 : taagged_recs) {
         if (tagged_rec.tag != tagged_rec1.tag) {
             joined_rec = join(tagged_rec, tagged_rec1)
         }
       emit (tagged_rec.rec.Dept_Id, joined_rec)
    }
}
```

http://www.javacodegeeks.com/2012/05/joins-with-map-reduce.html

Join (Hive QL)

```
INSERT OVERWRITE TABLE pv_users
SELECT pv.pageid, u.age_bkt
FROM page_view pv
 JOIN user u
 ON (pv.uhash = u.uhash);
```

Hive QL – Join in Map Reduce



Reduce Side Join

- Similar to nested-loop join
- Values from each group have the same join attribute => don't check the join attribute in the nested loop
- Check which relation each tuple comes from, so that we don't join a tuple from *R* with another tuple from *R*
- Shuffling of all data across the network, even though not matching records are later dropped => very inefficient

Select tuples from R: $\sigma_{a<10}R$

```
map(inkey, invalue)
   if inkey < 10
      emit_intermediate(inkey, invalue)


reduce(hkey, hvalues[])
   for each t in hvalues
       emit(t)
```

Eliminate duplicates from R: *δ(R)*

- Duplicate elimination in the bag relational algebra is equivalent to grouping on all attributes of the relation.
- MapReduce does grouping for us, so all we need is to make the entire tuple the intermediate key to group on.

Eliminate duplicates from R: *δ(R)*

```
map(inkey, invalue)
  // We won't use the intermediate value,
  // so we just put in a dummy value
  emit_intermediate(invalue, 'abc')

reduce(hkey, hvalues[])
  emit(hkey)
```

MapReduce

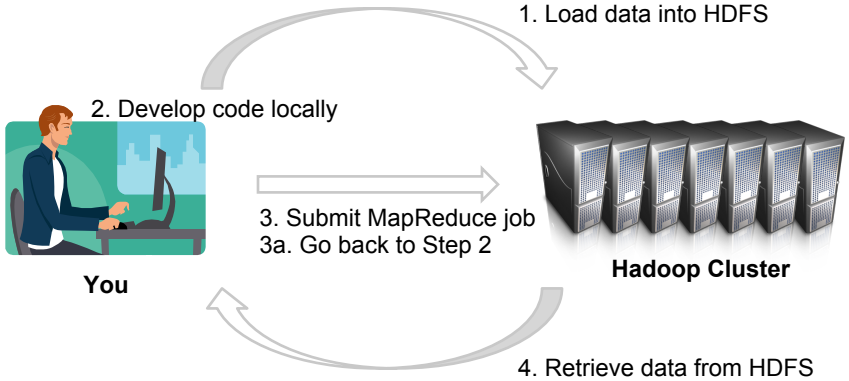
- Programmers specify the map and reduce functions
- The execution framework handles everything else…

What's "everything else"?

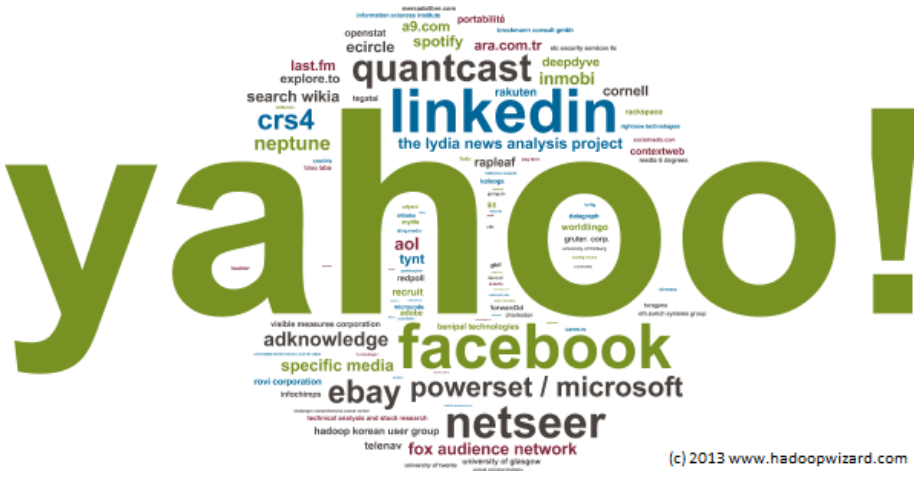
MapReduce "Runtime"

- Handles scheduling
  - Assigns workers to map and reduce tasks
  - Speculative execution ("stragglers")
- Handles "data distribution"
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed file system (DFS)

Hadoop Workflow



Current Status of Hadoop



Wordcloud of Companies by their Relative Hadoop Cluster Sizes, Number of Nodes, Dec. 2012

http://www.hadoopwizard.com/which-big-data-company-has-the-worlds-biggest-hadoop-cluster/

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
  - Hive: HQL is like SQL (declarative, flexible schema)
  - Pig: Pig Latin is a bit like Perl (imperative, with relational operators)
  - Both compile to "workflow" of Hadoop (MapReduce) jobs

Hive Overview

- What's Hive
  - A data warehousing system to store structured data on Hadoop file system
  - Provide an easy way to query this data by executing Hadoop MapReduce plans
- Intuitive
  - Make the unstructured data look like tables
  - SQL like queries used to query these tables
  - Generate specific execution plans for queries

HIVE

- Components
  - MapReduce for execution
  - HDFS for storage
  - Metadata in an RDBMS
- Key Building Principles
  - SQL as a familiar data warehousing tool
  - Extensibility – Types, Functions, Formats, Scripts
  - Scalability and Performance
  - Interoperability

Hive: Background

- Started at Facebook
- Data was collected by nightly cron jobs into Oracle DB
- "ETL" (Extract-Transform-Load) via hand-coded python
- Grew from 10s of GBs (2006) to 1 TB/ day new data (2007) to 100 PB (June 2012) and roughly half a PB per day (Nov 2012).

Source: cc-licensed slide by Cloudera

Why SQL on Hadoop?

```
hive> select key, count(1) from kv1 where key > 100 group
   by key;
```

vs.

```
$ cat > /tmp/reducer.sh
uniq -c | awk '{print $2"\t"$1}'
$ cat > /tmp/map.sh
awk -F '\001' '{if($1 > 100) print $1}'
$ bin/hadoop jar contrib/hadoop-0.19.2-dev-streaming.jar -input /user/hive/
    warehouse/kv1 -mapper map.sh -file /tmp/reducer.sh -file /tmp/map.sh -
    reducer reducer.sh -output /tmp/largekey -numReduceTasks 1
$ bin/hadoop dfs –cat /tmp/largekey/part*
```

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from Homer

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN homer k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985
to	18038	13526
of	16700	34654
a	14170	8057
you	12702	2720
my	11297	4135
in	10797	12445
is	8882	6884

Source: Material drawn from Cloudera training VM

Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN homer k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

(Abstract Syntax Tree)

(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF homer k) (= (. (TOK_TABLE_OR_COL s) word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k) freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))

(one or more of MapReduce jobs)

Hive Query Language

- SQL
  - Sub-queries in from clause
  - Equi-joins (including Outer joins)
  - Multi-table Insert
  - Multi-group-by
  - Embedding Custom Map/Reduce in SQL
- Sampling
- Primitive Types
  - integer types, float, string, boolean
- Nestable Collections
  - array<any-type> and map<primitive-type, any-type>
- User-defined types
  - Structures with attributes which can be of any-type

Hive & Hadoop Usage @ Facebook

- **Statistics per day:**
  - 12 TB of compressed new data added per day
  - 135TB of compressed data scanned per day
  - 7500+ Hive jobs per day
  - 80K compute hours per day
- **Hive simplifies Hadoop:**
  - New engineers go though a Hive training session
  - ~200 people/month run jobs on Hadoop/Hive
  - Analysts (non-engineers) use Hadoop through Hive
  - 95% of jobs are Hive Jobs

Hive & Hadoop Usage @ Facebook

- Types of Applications:
  - Reporting
    - Eg: Daily/Weekly aggregations of impression/click counts
    - Measures of user engagement
    - Microstrategy reports
  - Ad hoc Analysis
    - Eg: how many group admins broken down by state/country
  - Machine Learning (Assembling training data)
    - Ad Optimization
    - Eg: User Engagement as a function of user attributes
  - Many others

More Real-World Use Cases

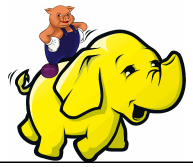
- Bizo: Use Hive for reporting and ad hoc queries.
- Chitika: … for data mining and analysis …
- CNET: … for data mining, log analysis and ad hoc queries
- Digg: … data mining, log analysis, R&D, reporting/analytics
- Grooveshark: … user analytics, dataset cleaning, machine learning R&D.
- Hi5: … analytics, machine learning, social graph analysis.
- HubSpot: … to serve near real-time web analytics.
- Last.fm: … for various ad hoc queries.
- Trending Topics: … for log data normalization and building sample data sets for trend detection R&D.
- VideoEgg: … analyze all the usage data

HIVE - Conclusions

- A easy way to process large scale data.
- Support SQL-based queries.
- Provide more user defined interfaces to extend programmability.

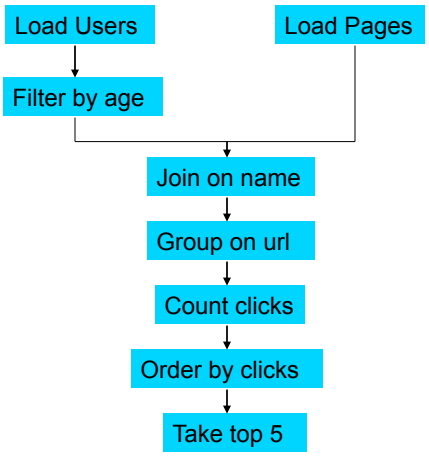
Need for High-Level Languages

- 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
  - Hive: HQL is like SQL (declarative, flexible schema)
  - Pig: Pig Latin is a bit like Perl (imperative, with relational operators)
  - Both compile to "workflow" of Hadoop (MapReduce) jobs



An Example Problem

- Data
  - User records
  - Pages served
- Question: the 5 pages most visited by users aged 18 - 25.



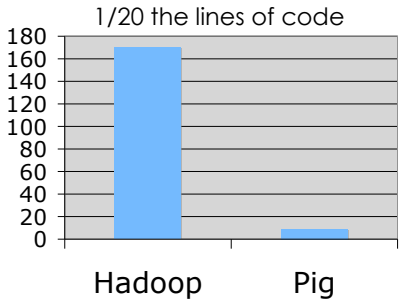
In Map Reduce

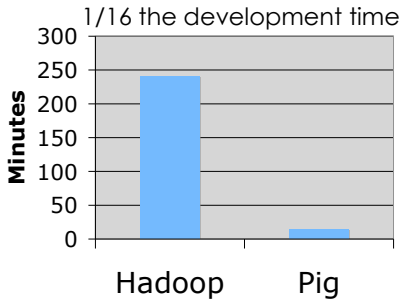
```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

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(IdentityMapper.class);
        lp.setReducerClass(IdentityMapper.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';
```

Comparison





Pig Compared to Map Reduce

- Faster development time
- Data flow versus programming logic
- Many standard data operations (e.g. join) included
- Manages all the details of connecting jobs and data flow
- Copes with Hadoop version change issues

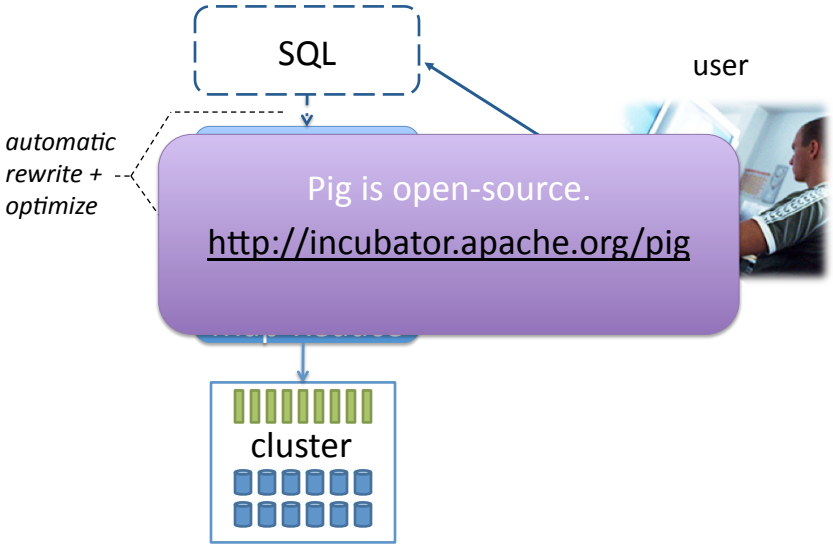
And, You Don't Lose Power

- UDFs can be used to load, evaluate, aggregate, and store data
- External binaries can be invoked
- Metadata is optional
- Flexible data model
- Nested data types
- Explicit data flow programming

Pig Commands

load	Read data from file system.
store	Write data to file system.
foreach	Apply expression to each record and output one or more records.
filter	Apply predicate and remove records that do not return true.
group/cogroup	Collect records with the same key from one or more inputs.
join	Join two or more inputs based on a key. Various join algorithms available.
order	Sort records based on a key.
distinct	Remove duplicate records.
union	Merge two data sets.
split	Split data into 2 or more sets, based on filter conditions.
stream	Send all records through a user provided executable.
sample	Read a random sample of the data.
limit	Limit the number of records.

Implementation



What Users Do with Pig

- Inside Yahoo (based on user interviews)
  - 60% of ad hoc and 40% of production MR jobs
  - Production
    - Examples: search infrastructure, ad relevance
    - Attraction: fast development, extensibility via custom code, protection against Hadoop changes, debugability
  - Ad hoc
    - Examples: user intent analysis
    - Attraction: easy to learn, compact readable code, fast iteration when trying new algorithms, easy for collaboration

What Users Do with Pig

- Outside Yahoo (based on mailing list responses)
  - Processing search engine query logs
    "Pig programs are easier to maintain, and less error-prone than native java programs. It is an excellent piece of work."
  - Image recommendations
    "I am using it as a rapid-prototyping language to test some algorithms on huge amounts of data."
  - Adsorption Algorithm (video recommendations)
  - Hoffman's PLSI implementation
    "The E/M login was implemented in pig in 30-35 lines of pig-latin statements. Took a lot less compared to what it took in implementing the algorithm in mapreduce java. Exactly that's the reason I wanted to try it out in Pig. It took ~ 3-4 days for me to write it, starting from learning pig."

Hive versus Pig

Feature	Hive	Pig
Language	SQL-like	PigLatin
Schemas/Types	Yes (explicit)	Yes (implicit)
Partitions	Yes	No
Server	Optional (Thrift)	No
User Defined Functions (UDF)	Yes (Java)	Yes (Java)
Custom Serializer/Deserializer	Yes	Yes
DFS Direct Access	Yes (implicit)	Yes (explicit)
Join/Order/Sort	Yes	Yes
Shell	Yes	Yes
Streaming	Yes	Yes
Web Interface	Yes	No
JDBC/ODBC	Yes (limited)	No

http://www.larsgeorge.com/2009/10/hive-vs-pig.html

Summary

Hive:

- Helpful for ETL
- Very good for Ad-Hoc Analysis - Not necessarily suited for front end users but definitely helpful for data analysts
- Directly leverages SQL expertise!!

PIG:

- Great for ETL
- Powerful, transformation and processing capabilities
- SQL-like, but different in many ways, will take some time to master.