Pig, a high level data processing system on Hadoop

Is MapReduce not Good Enough?

- Restricted programming model
 - Only two phases
 - Job chain for long data flow
- Too many lines of code even for simple logic
 - How many lines do you have for word count?
 - Programmers are responsible for this

Pig to the Rescue

- High level dataflow language (Pig Latin)
 - Much simpler than Java
 - Simplifies the data processing
- Puts the operations at the apropriate phases
- Chains multiple MR jobs

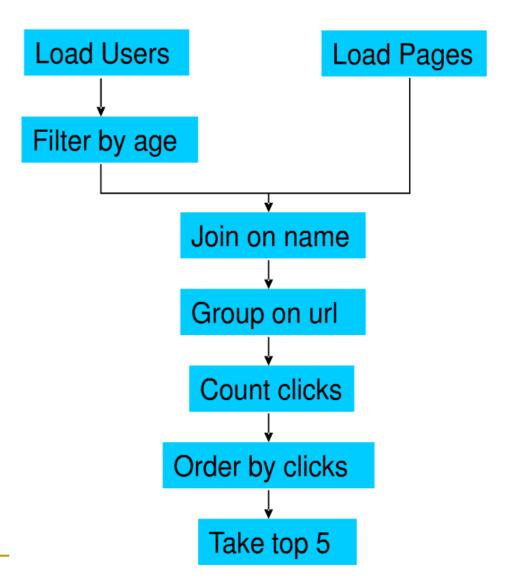
How Pig is used in the Industry

- At Yahoo, 70% MapReduce jobs are written in Pig
- Used to
 - Process web logs
 - Build user behavior models
 - Process images
 - Data mining

Also used by Twitter, LinkedIn, eBay, AOL, ...

Motivation by Example

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



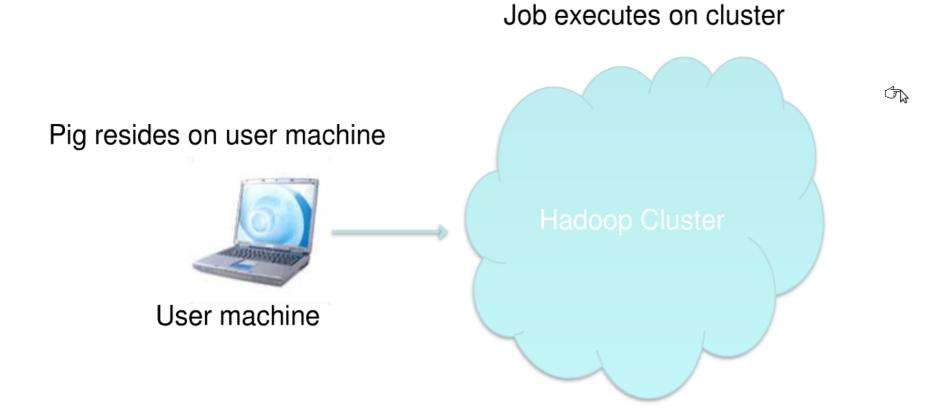
In MapReduce

import java.io.ICException; reporter.setStatus("OK"); lp.setOutputKevClass(Text.class); import java.util.ArrayList; import java.util.Iterator; import java.util.List; - 5 lp.setOutputValueClass(Text.class); lp.setMapperClass(LoadPages.class); FileInputFormat.addInputPath(lp, new // Do the cross product and collect the values for (String s1 : first) (for (String s2 : second) { String outval = key + "," + s1 + "," + oc.collect(null, new Text(outval)); reporter.setStatus("CK"); Path (d's/er/gates/pages")); FileOutputFormat.setOutputPath(1p, import org.apache.hadoop.fs.Path; import org.apache.hadoop.rs.Path; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text; s2; new Path("/user/gates/tmp/indexed_p
lp.setNumReduceTasks(0); Job loadPages = new Job(lp); import org.apache.hadoop.io.Writable; reporter.setStatus("CK");
import org.apache.hadoop.io.WritableComparable; }
import org.apache.hadoop.mapred.FileInputFormat; }
import org.apache.hadoop.mapred.FileCutputFormat; }
import org.apache.hadoop.mapred.JobConf; }
import org.apache.hadoop.mapred.KeyValueTextInputFormat;public static class LoadJoined extends MapReduceBase
import org.apache.hadoop.mapred.Mapper; implements Mapper/Text, Text, Text, LongWritable>
import org.apache.hadoop.mapred.MapReduceBase; implements Mapper/Text, Text, Text, LongWritable> JobConf lfu = new JobConf(MRExample.cla lafuJambName("Load and Filter Users"); lfu.setInputFormat.Clas lfu.setOutputKeyClass(Text.class); fu.setOutputValueClass(Text.cla lfu.setMapperClass(LoadAndFilterUsers.c import org.apache.hadoop.mapred.CutputCollector; import org.apache.hadoop.mapred.RecordReader; public void map(FileInputFårmatPaddd(lfu, new user/gates/users")); Path ("/ Text k, import org.apache.hadoop.mapred.Reducer; import org.apache.hadoop.mapred.Reporter; Text val. Cutpotesate, LongWritable> oc, // Reporter reporter) throws IOException (
// Find the url
String line = val.toString(); imprt org.apache.hadoop.mapred.SequenceFileInputFormat; import org.apache.hadoop.mapred.SequenceFileCutputFormat; import org.apache.hadoop.mapred.TextInputFormat; import org.apache.hadoop.mapred.jobcontrol.Job; import org.apache.hadoop.mapred.jobcodntrol.Jobc import org.apache.hadoop.mapred.lib.IdentityMapper; public class MRExample (lic static class LoadPages extends MapReduceBase implements Mapper<LongWritable, Text, Text, Text> { public public void map{LongWritable k, Text val, OutputCollector<Text, Text> oc, OutputCollector<Text, Text> oc, public static class ReduceUrls extends MapReduceBase FilefnputFormat.addInputFormat.ddInput JODCONT group = newsupsecontdMsk; group.setJobName("Group URLs"); group.setInputFormat(KeyValueTextInputF group.setCutputKeyClass(Text.class); group.setOutputValueClass(LongWritable. public static class LoadAndFilterUsers extends MapReduceBase implements Mapper<LongWritable, Text, Text, Text> { } group.setCutputFormlactC(UStepguteRhomenFait.class group.setMapperClass(LoadJoined.class) group.setCombinerClass (ReduceUrls, class public void map(LongWritable k, Text val, CutputCollector<Text, Text> oc, Reporter reporter) throws IOException { } // pull the key out String line = val.toString{}, group.setCombinerClass(ReduceUrls.class)
group.setReducerClass(ReduceUrls.class)
FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
eBase FileOutputFormat.setOutputPath(group, new oc.collect(key, new LongWritable(sum)); mplements Mapper<WritableComparable, Writable,FaltonigW/ruisteaub/loga,tes/tmp/grouped")); Text> (public static class LoadClicks extends MapReduceBase int firstComma = line.indexOf(',');
String value = lfmestCommaing(); group.setNumReduceTasks(50); Job groupJob = new Job(group); string value = linessCommring(); int age = Integer.parseInt(value); if (age < 18 || age > 25) return; String (ey = line.substring (0, firstComma); Text outKey = new Text(key); // Prepend an index too knows wahluceh sfoilwe // it came from. Text outVal = new Text("2" + value); oc.collect(outKey, outVal); public void map(WritableComparable key, groupJob,addDependingJob(joinJob); JobConf top100 = new JobConf(MRExample. Writable val, OutputCollector<LongWritable, Text> oc, LobConf top100 = new LobConf (MKExample. top100.setJobName("Top 100 sites"); top100.setInputFormat (SequenceFileInput top100.setGutputKeyClass(LongWritable.c top100.setGutputValueClass(Text.class); ReportenthmomogrueException { oc.collect((LongWritable)val, (Text)key); 3 top100.setCutputValueclass(rext.class);
 top100.setCutputValueclass(SequrmataRilbacQutpu
public static class LimitClicks extends MapReduceBase top100.setMapperClass(LoadClicks.class);
 implements Reducer<LongWritable, Text, LongWritable,tdDputValueclass(LimitClicks.class);
 top100.setReducerClass(LimitClicks.class);
 top100.setReducerClass(LimitClicks);
 top100.setReducerClas . public static class Join extends MapReduceBase FileInputFormat.addInputPath(top100, ne
Path("/user/gates/tmp/grouped"));
FileOutputFormat.setOutputPath(top100, ne
Path("/user/gates/top100sitesforusers18to25"));
top100.setNumReduceTasks(1); implements Reducer<Text, Text, Text, Text> (int count = 0; Int count = v; pub/bidd reduce(LongWritable key, Iterator(Text> iter, GutputCollector(LongWritable, Text> oc, Reporter reporter) throws IOException (public void reduce(Text key, Iterator<Text, Key, Iterator<Text> iter, CutputCollector<Text, Text> oc, Reporter reporter) throws IOException { Rep // For each value, figure out which file it's from and Job limit = new Job(top100); limit.addDependingJob(groupJob); store it // accordingly. List<String> first = new ArrayList<String>{}; List<String> second = new ArrayList<String>{}; JobControl je = new JobCohû@oùit#sindoto 18 to 25"); jc.addJob(loadPages); jc.addJob(loadUsers); while (iter.hasNext()) (
 Text t = iter.next();
 String Wsthue.ge()t.to
 if (value.charAt(0) == '1')
first.add(value.substring(1)); jc.addJob(joinJob); c.addJob(groupJob); else second, add (value, substring(1)); lp.setInputFormat(TextInputFormat.class); }

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 = joinFltrdby name, Pages by user;
Grpd = groupJndbyurl;
Smmd = foreachGrpdgenerate group,
COUNT (Jnd) as clicks;
Srtd = orderSmmdby clicks desc;
Top5 = limitSrtd 5;
store Top5 into `top5sites';
```

Pig runs over Hadoop



No need to install anything extra on your Hadoop cluster.

Wait a minute

How to map the data to records

- $\hfill\square$ By default, one line \rightarrow one record
- User can customize the loading process
- How to identify attributes and map them to the schema
 - Delimiter to separate different attributes
 - By default, delimiter is tab. Customizable.

MapReduce Vs. Pig cont.

- Join in MapReduce
 - Various algorithms. None of them are easy to implement in MapReduce
 - Multi-way join is more complicated
 - Hard to integrate into SPJA workflow

MapReduce Vs. Pig cont.

Join in Pig

- Various algorithms are already available.
- Some of them are generic to support multi-way join
- No need to consider integration into SPJA workflow. Pig does that for you!

Pig Latin

- Data flow language
 - Users specify a sequence of operations to process data
 - More control on the process, compared with declarative language
- Various data types are supported
- Schema is supported
- User-defined functions are supported

Statement

- A statement represents an operation, or a stage in the data flow
- Usually a variable is used to represent the result of the statement
- Not limited to data processing operations, but also contains filesystem operations

Schema

- User can optionally define the schema of the input data
- Once the schema of the source data is given, the schema of the intermediate relation will be induced by Pig

Schema cont.

- Why schema?
 - Scripts are more readable (by alias)
 - Help system validate the input
- Similar to Database?
 - Yes. But schema here is optional
 - Schema is not fixed for a particular dataset, but changable

Schema cont.

Schema 1

A = LOAD 'input/A' as (name:chararray, age:int);

B = FILTER A BY age != 20;

Schema 2

A = LOAD 'input/A' as (name:chararray, age:chararray);

B = FILTER A BY age != '20';

No Schema

A = LOAD 'input/A'; $B = EII TEP \land BY \land $1 = 2$

B = FILTER A BY A. 1 != '20';

Data Types

- Every attribute can always be interpreted as a bytearray, without further type definition
- Simple data types
 - For each attribute
 - Defined by user in the schema
 - □ Int, double, chararray ...
- Complex data types
 - Usually contructed by relational operations
 - Tuple, bag, map

Data Types cont.

- Type casting
 - Pig will try to cast data types when type inconsistency is seen.
 - Warning will be thrown if casting fails. Process still goes on
- Validation
 - Null will replace the inconvertable data type in type casting
 - User can tell a corrupted record by detecting whether a particular attribute is null

Date Types cont.

1950	0	1	<pre>grunt> records = LOAD 'input/ncdc/micro-tab/sample_corrupt.txt' >> AS (year:chararray, temperature:int, quality:int); grunt> DUMP records; (1950,0,1) (1950,22,1) (1950,,1) (1949,111,1) (1949,78,1)</pre>
1950	22	1	
1950	е	1	
1949	111	1	
1949	78	1	

```
grunt> corrupt_records = FILTER records BY temperature is null;
grunt> DUMP corrupt_records;
(1950,,1)
```

Operators

- Relational Operators
 - Represent an operation that will be added to the logical plan
 - LOAD, STORE, FILTER, JOIN, FOREACH...GENERATE

```
arunts DIMD A.
grunt> DUMP A;
(2, Tie)
                            grunt> C = JOIN A BY $0, B BY $1;
(4,Coat)
                            grunt> DUMP C;
(3,Hat)
                             (2, Tie, Joe, 2)
(1,Scarf)
                  EACH A G
                             (2,Tie,Hank,2)
grunt> DUMP B;
                             (3, Hat, Eve, 3)
(Joe,2)
                  t)
t)
t)
                             (4,Coat,Hank,4)
(Hank, 4)
(Ali,0)
(Eve,3)
                  t)
(Hank,2)
```

Operators

- Diagnostic Operators
 - Show the status/metadata of the relations
 - Used for debugging
 - Will not be integrated into execution plan
 - DESCRIBE, EXPLAIN, ILLUSTRATE.

```
grunt> records = LOAD 'input/ncdc/micro-tab/sample.txt'
>> AS (year, temperature:int, quality:int);
grunt> DESCRIBE records;
records: {year: bytearray,temperature: int,quality: int}
```

Functions

- Eval Functions
 - Record transformation
- Filter Functions
 - Test whether a record satisfies particular predicate
- Comparison Functions
 - Impose ordering between two records. Used by ORDER operation
- Load Functions
 - Specify how to load data into relations
- Store Functions
 - Specify how to store relations to external storage

Functions

- Built-in Functions
 - Hard-coded routines offered by Pig.
- User Defined Function (UDF)
 - Supports customized functionalities
 - Piggy Bank, a warehouse for UDFs

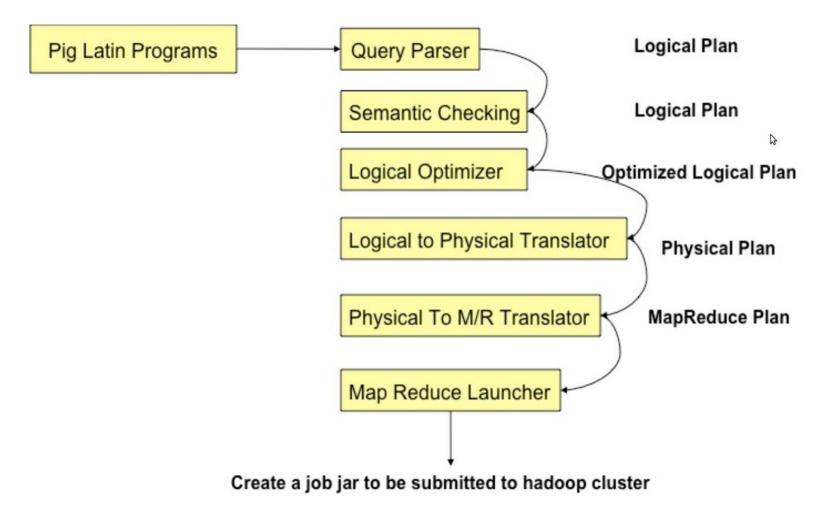
View of Pig from inside

Pig Execution Modes

Local mode

- Launch single JVM
- Access local file system
- No MR job running
- Hadoop mode
 - Execute a sequence of MR jobs
 - Pig interacts with Hadoop master node

Compilation



Parsing

- Type checking with schema
- Reference verification
- Logical plan generation
 - One-to-one fashion
 - Independent of execution platform
 - Limited optimization
 - No execution until DUMP or STORE

Logical Plan

A=LOAD 'file1' AS (x, y, z);

B=LOAD 'file2' AS (t, u, v);

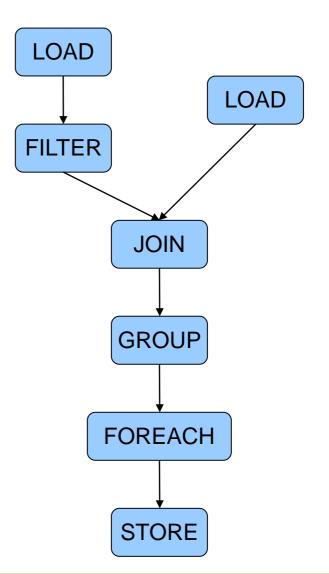
C=FILTER A by y > 0;

D=JOIN C BY x, B BY u;

E=GROUP D BY z;

F=FOREACH E GENERATE group, COUNT(D);

STORE F INTO 'output';



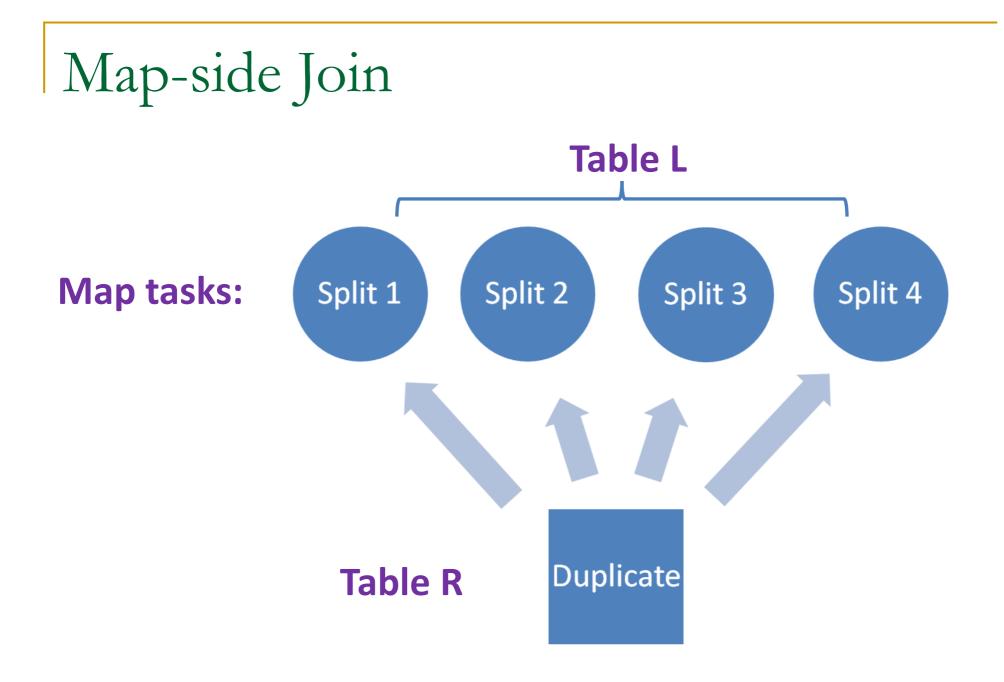
Physical Plan

- 1:1 correspondence with most logical operators
- Except for:
 - DISTINCT
 - (CO)GROUP
 - JOIN
 - ORDER

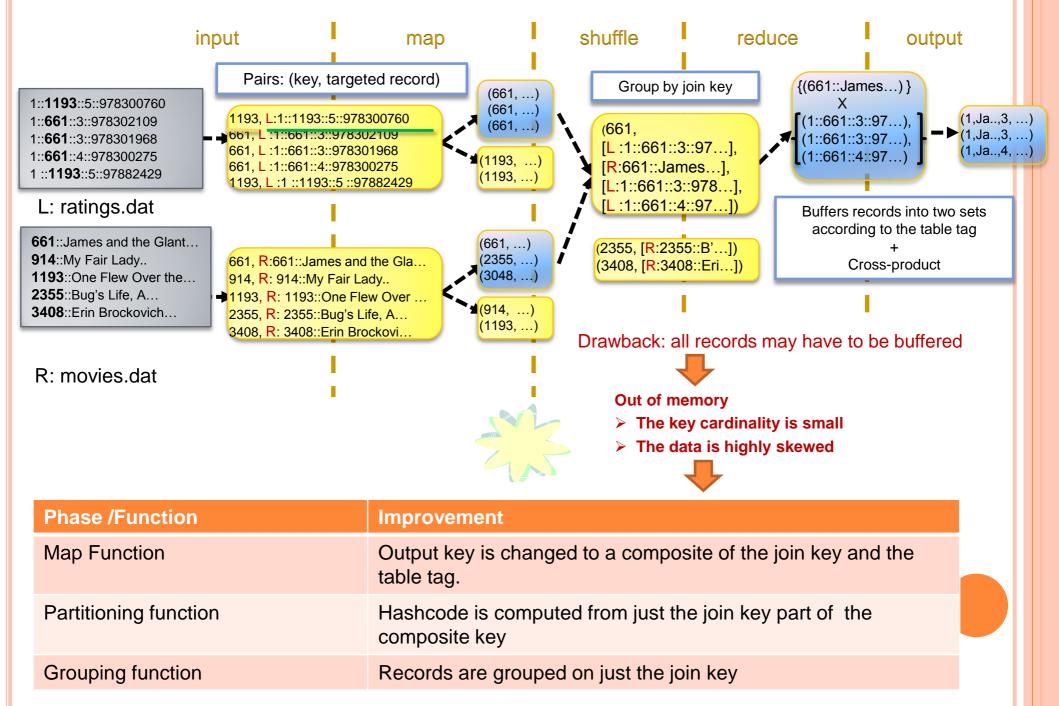
Joins in MapReduce

Two typical types of join

- Map-side join
- Reduce-side join

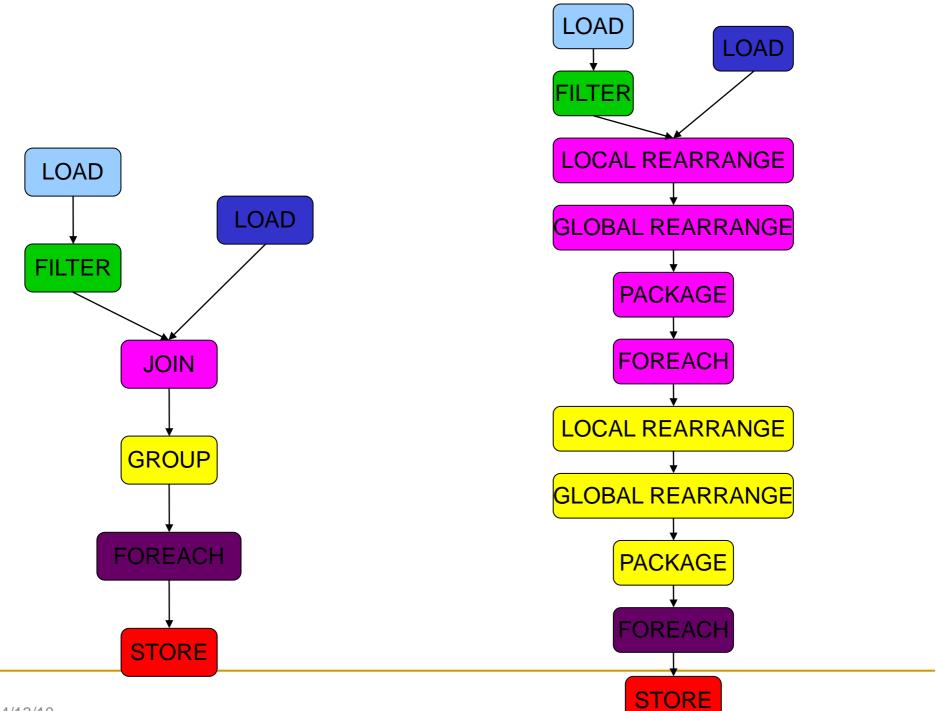


REDUCE-SIDE JOIN



Physical Plan

- 1:1 correspondence with most logical operators
- Except for:
 - DISTINCT
 - (CO)GROUP
 - JOIN
 - ORDER

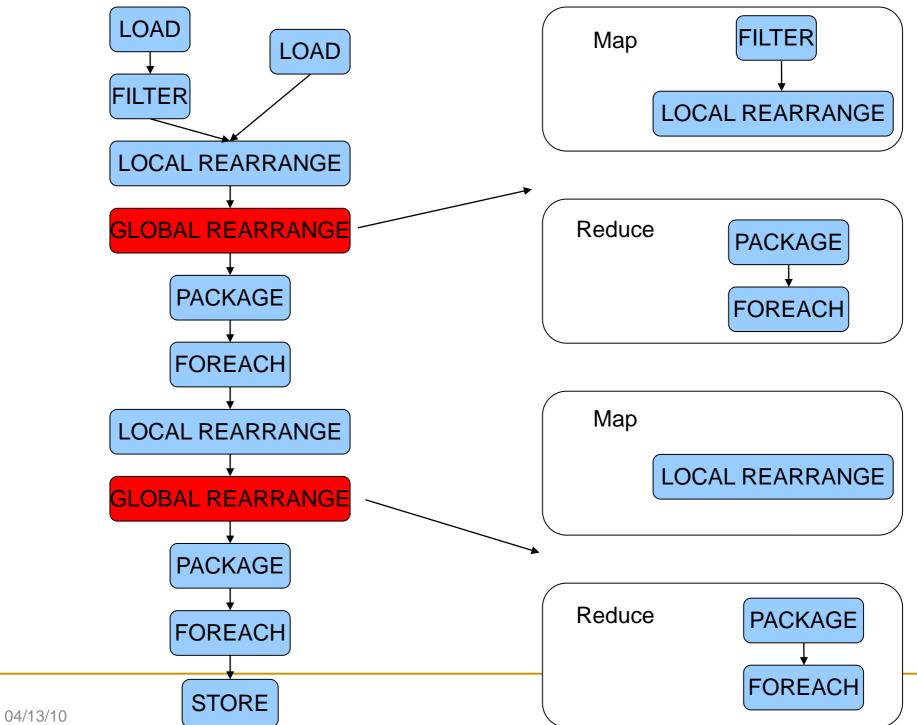


Physical Optimizations

- Always use combiner for pre-aggregation
- Insert SPLIT to re-use intermediate result
- Early projection (logical or physical?)

MapReduce Plan

- Determine MapReduce boundaries
 - GLOBAL REARRANGE
 - STORE/LOAD
- Some operations are done by MapReduce framework
- Coalesce other operators into Map & Reduce stages
- Generate job jar file



Execution in Hadoop Mode

- The MR jobs not dependent on anything in the MR plan will be submitted for execution
- MR jobs will be removed from MR plan after completion
 - Jobs whose dependencies are satisfied are now ready for execution
- Currently, no support for inter-job faulttolerance

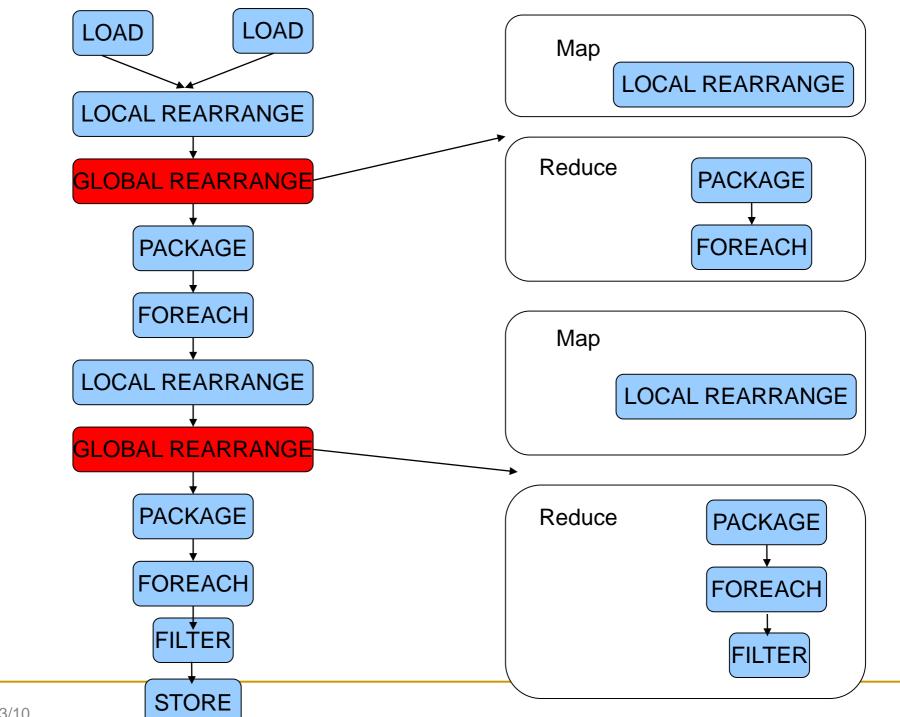
Discussion of the Two Readings on Pig (SIGMOD 2008 and VLDB 2009)

Discussion Points for Reading 1

- Examples of the nested data model, CoGroup, and Join (Figure 2)
- Nested query in Section 3.7

What are the Logical, Physical, and MapReduce plans for:

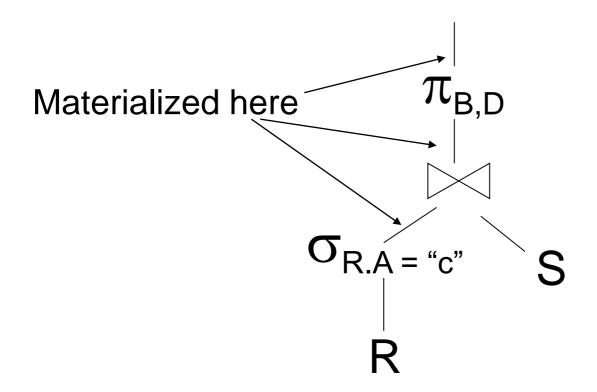
Operators LOAD GROUP COGROUP FILTER FOREACH ORDER				
□ = LOAD USING Default ✓ AS () Generate Query				
visits = LOAD 'visits.txt' AS (user, url, time);	visits:	(Amy, cnn.com, 8am) (Amy, frogs.com, 9am) (Fred, snails.com, 11am)		
pages = LOAD 'pages.txt' AS (url, pagerank);	pages:	(cnn.com, 0.8) (frogs.com, 0.8) (snails.com, 0.3)		
v_p = JOIN visits BY url, pages BY url;	v_p:	(Amy, cnn.com, 8am, cnn.com, 0.8) (Amy, frogs.com, 9am, frogs.com, 0.8) (Fred, snails.com, 11am, snails.com, 0.3)		
users = GROUP v_p BY user;	users:	<pre>(Amy, { (Amy, cnn.com, 8am, cnn.com, 0.8),</pre>		
useravg = FOREACH users GENERATE group, AVG(v_p.pagerank) AS avgpr;	useravg:	(Amy, 0.8) (Fred, 0.3)		
answer = FILTER useravg BY avgpr > '0.5';	answer:	(Amy, 0.8)		
STORE answer INTO '/user/alan/answer';				



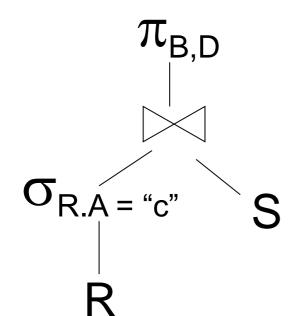
Recall Operator Plumbing $\pi_{B,D}$ $\sigma_{R,A="c"}$ S

- Materialization: output of one operator written to disk, next operator reads from the disk
- Pipelining: output of one operator directly fed to next operator

Materialization



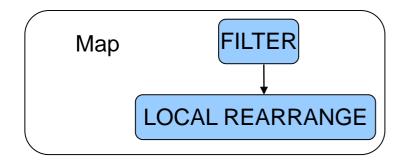
Iterators: Pipelining

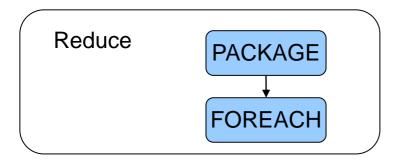


- → Each operator supports:
 - Open()
 - GetNext()
 - Close()

How do these operators execute in Pig?

1950	0	1
1950	22	1
1950	е	1
1949	111	1
1949	78	1





- Hints (based on Reading 2):
 - What will Hadoop's map function and reduce function calls do in this case?
 - How does each operator work?
 What does each operator do?
 (Section 4.3)
 - Outermost operator graph (Section 5)
 - Iterator model (Section 5)

Branching Flows in Pig

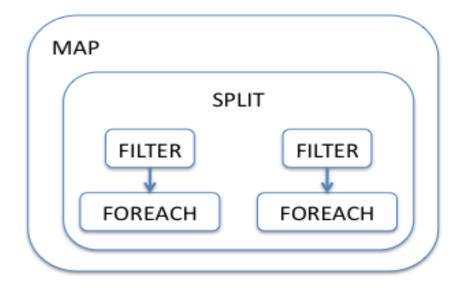
clicks = LOAD `clicks' AS (userid, pageid, linkid, viewedat);

SPLIT clicks INTO pages IF pageid IS NOT NULL, links IF linkid IS NOT NULL;

```
cpages = FOREACH pages GENERATE userid,
CanonicalizePage(pageid) AS cpage,
viewedat;
```

clinks = FOREACH links GENERATE userid, CanonicalizeLink(linkid) AS clink, viewedat;

STORE cpages INTO `pages'; STORE clinks INTO `links';



- Hints (based on Reading 2, Section 5.1, last two paras before Section 5.1.1):
 - Outermost data flow graph
 - New pause signal for iterators

Branching Flows in Pig

Draw the MapReduce plan for this query

clicks = LOAD `clicks' AS (userid, pageid, linkid, viewedat);

byuser = GROUP clicks BY userid;

result = FOREACH byuser {

uniqPages = DISTINCT clicks.pageid;

uniqLinks = DISTINCT clicks.linkid;

GENERATE group, COUNT(uniqPages), COUNT(uniqLinks);

};

Branching Flows in Pig

Draw the MapReduce plan for this query

clicks = LOAD `clicks' AS (userid, pageid, linkid, viewedat);

byuser = GROUP clicks BY userid;

result = FOREACH byuser {

fltrd = FILTER clicks BY viewedat IS NOT NULL;

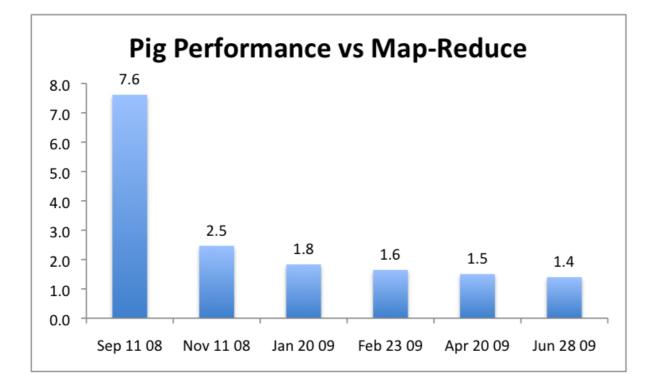
uniqPages = DISTINCT fltrd.pageid;

uniqLinks = DISTINCT fltrd.linkid;

```
GENERATE group, COUNT(uniqPages),
COUNT(uniqLinks);
};
```

Performance and future improvement

Pig Performance



Images from http://wiki.apache.org/pig/PigTalksPapers

Future Improvements

Query optimization

- Currently rule-based optimizer for plan rearrangement and join selection
- Cost-based in the future
- Non-Java UDFs
- Grouping and joining on pre-partitioned/sorted data
 - Avoid data shuffling for grouping and joining
 - Building metadata facilities to keep track of data layout
- Skew handling
 - For load balancing

- Get more information at the Pig website
- You can work with the source code to implement something new in Pig
- Also take a look at Hive, a similar system from Facebook

References

- Some of the content come from the following presentations:
 - Introduction to data processing using Hadoop and Pig, by Ricardo Varela
 - □ Pig, Making Hadoop Easy, by Alan F. Gates
 - Large-scale social media analysis with Hadoop, by Jake Hofman
 - Getting Started on Hadoop, by Paco Nathan
 - MapReduce Online, by Tyson Condie and Neil Conway