Advanced Pig Programming

2:30-3:30pm

Agenda

- **PIG Internals**
  - Logical Physical and M/R plan construction
  - Multi-query optimization
- **Writing your own UDF's**
  - **Eval Function**
  - **Filter Function**
  - **Accumulator Interface**
- **Zebra and Pig**

Pig Latin = Sweet Spot between SQL & Map-Reduce

	SQL	Pig	Map-Reduce
Programming style	Large blocks of declarative constraints	→	"Plug together pipes"
Built-in data manipulations	Group-by, Sort, Join, Filter, Aggregate, Top-k, etc...	←	Group-by, Sort
Execution model	Fancy; trust the query optimizer	→	Simple, transparent
Opportunities for automatic optimization	Many	←	Few (logic buried in map() and reduce())
Data Schema	Must be known at table creation	→	Not required, may be defined at runtime

Pig Latin Program an Example

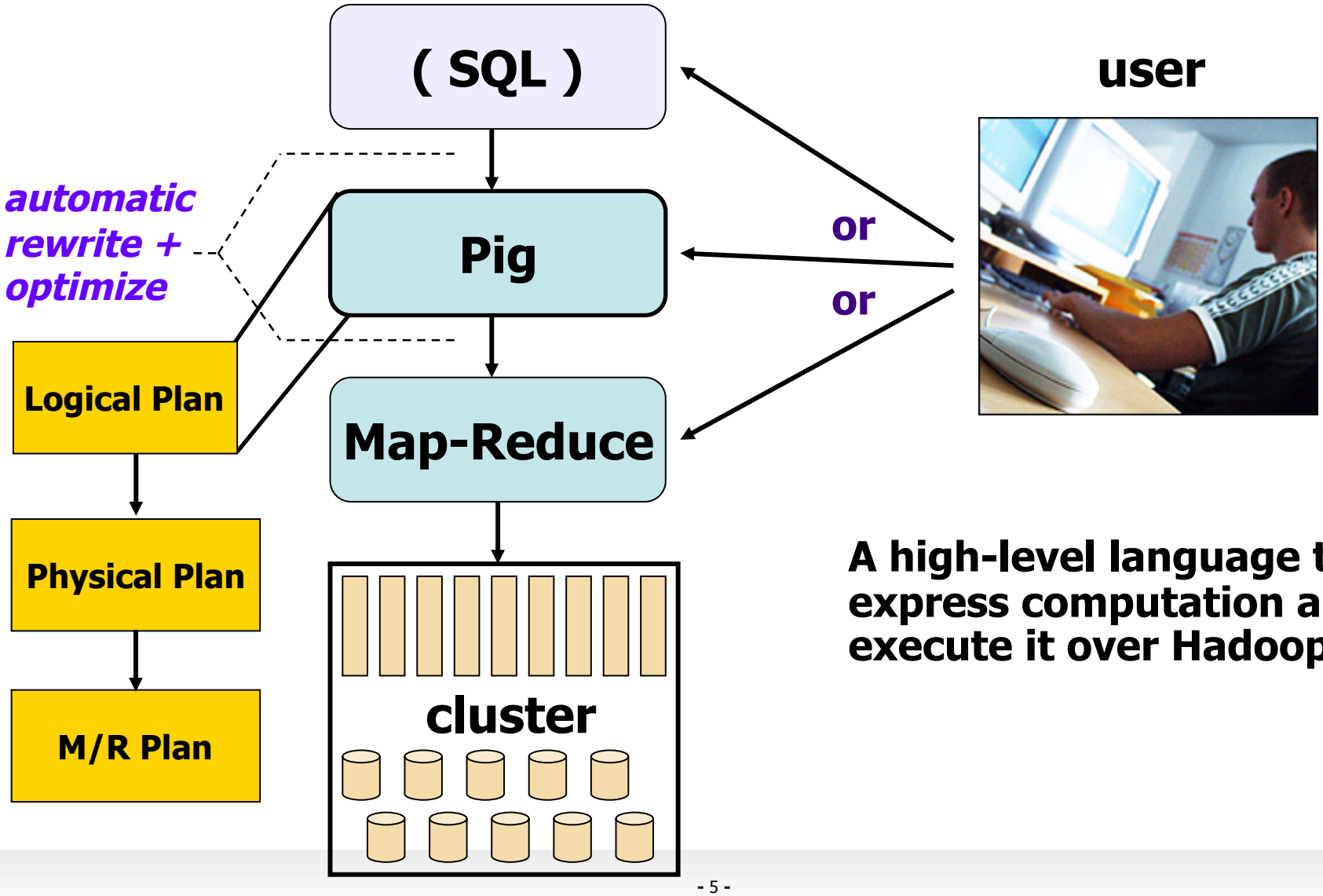
We have a dataset

```
urls: (url, category, pagerank)
```

We want to know the top 10 urls per category as measured by pagerank for sufficiently large categories:

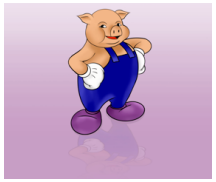
```
urls  = load 'dataset'
                    as (url, category, pagerank);
grps  = group urls by category;
bgrps = filter grps by
                    COUNT(urls) > 1000000;
rslt  = foreach bgrps
                    generate group, top10(urls);
store rslt into 'myOutput';
```

Pig Architecture: Map-Reduce as Backend



A high-level language to express computation and execute it over Hadoop

From Pig Latin to Map Reduce



Script

```
A = load
B = filter
C = group
D = foreach
```

Parser

Logical Plan ≈ relational algebra

Logical Plan

Semantic Checks

Logical Plan

Plan standard optimizations

Logical Optimizer

Logical Plan

Logical to Physical Translator

Physical Plan

Physical To MR Translator

Map-Reduce Plan

MapReduce Launcher

Jar to hadoop

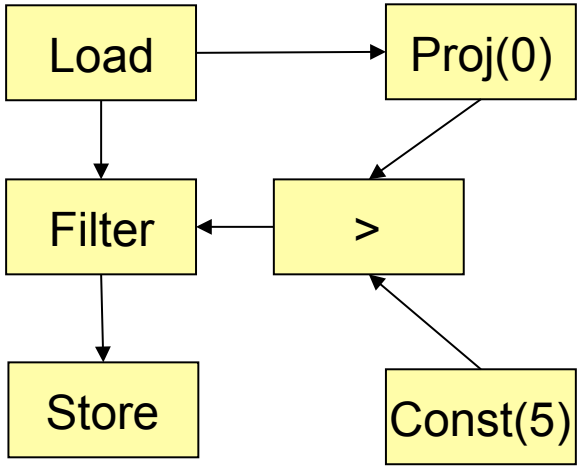
Map-Reduce Plan = physical operators broken into Map, Combine, and Reduce stages

Physical Plan = physical operators to be executed

Logical Plan

- Consists of DAG of Logical Operators as nodes and Data Flow represented as edges
  - Logical Operators contain list of i/p's o/p's and schema
- Logical operators
  - Aid in post parse stage checking (type checking)
  - Optimization
  - Translation to Physical Plan

```
a = load 'myfile';
b = filter a by $0 > 5;
store b into 'myfilteredfile';
```

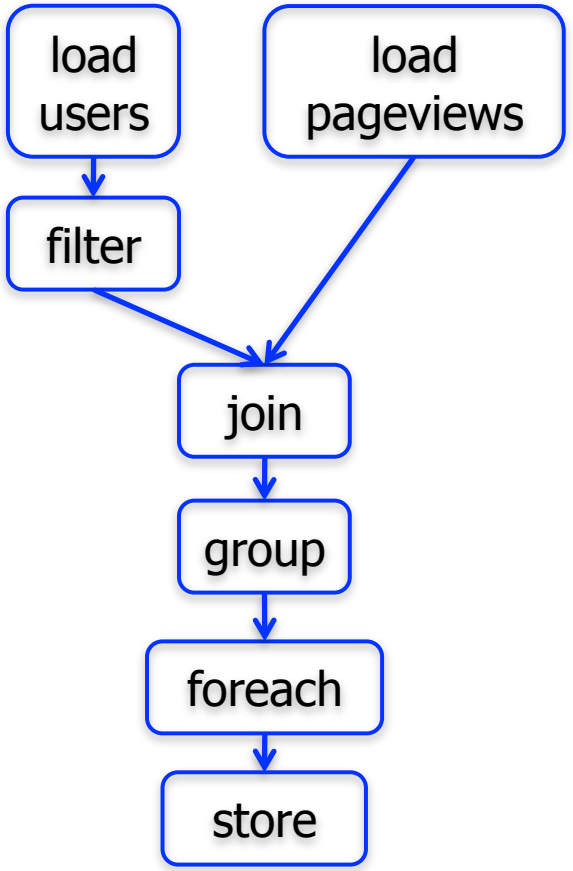


Pig Latin to Logical Plan

Pig Latin

```
A = load 'users' as (user, age);
B = load 'pageviews' as (user, url);
C = filter A by age < 18;
D = join A by user, B by user;
E = group D by url;
F = foreach E generate
        group, CalcScore(url);
store F into 'scored_urls';
```

Logical Plan

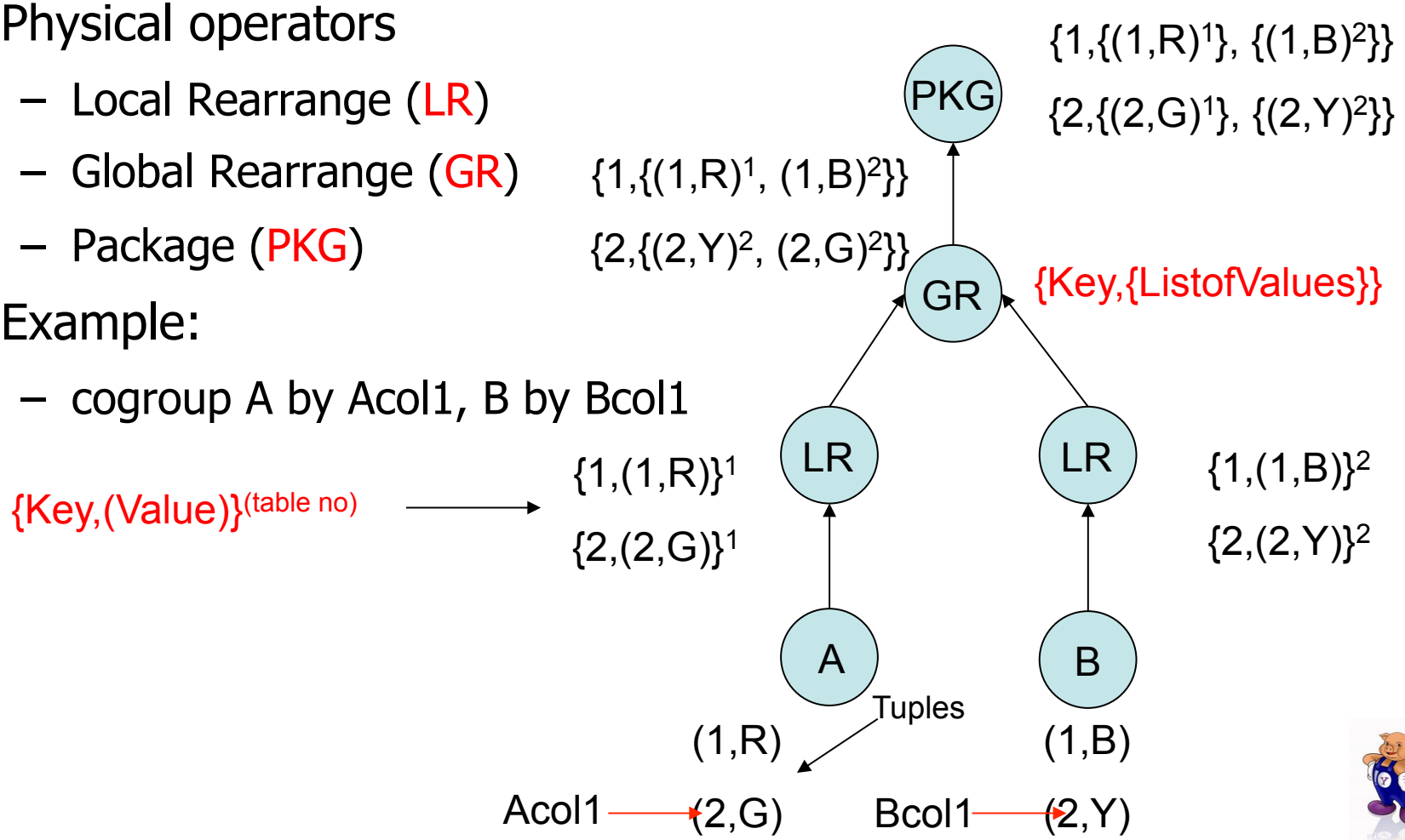


Physical Plan

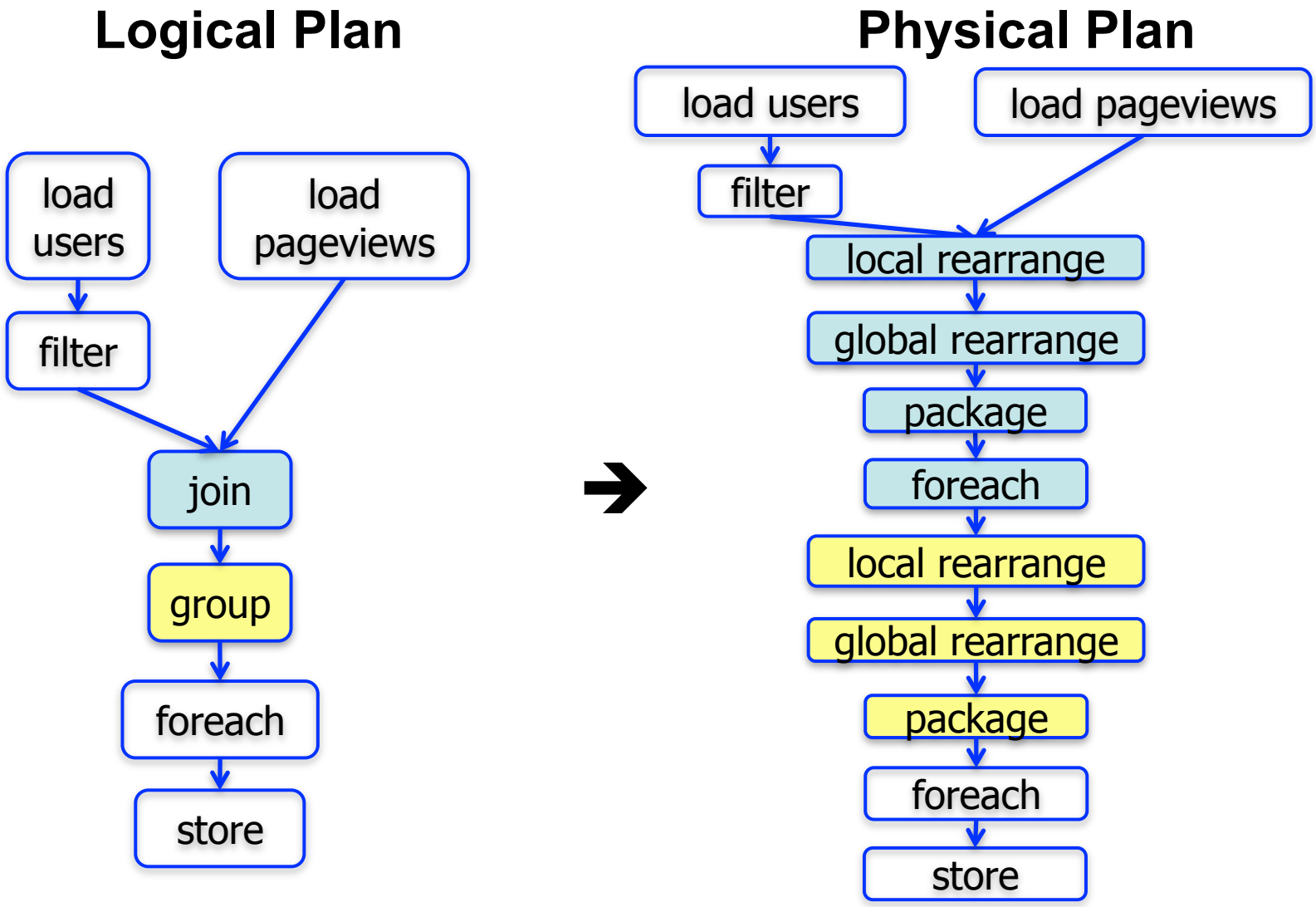
- Layer to map Logical Plan to multiple back-ends, one such being M/R (Map Reduce)
  - Chance for code re-use if multiple back-ends share same operator
- Consists of operators which Pig will run on the backend
- Currently most of the physical plan is placed as operators in the map reduce plan
- Logical to Physical Translation
  - 1:1 correspondence for most Logical operators
  - except Cross, Distinct, Group, Co-group and Order

Logical to Physical Plan for Co-Group operator

- Logical operator for co-group/group is converted to 3 Physical operators
  - Local Rearrange (LR)
  - Global Rearrange (GR)
  - Package (PKG)
- Example:
  - cogroup A by Acol1, B by Bcol1

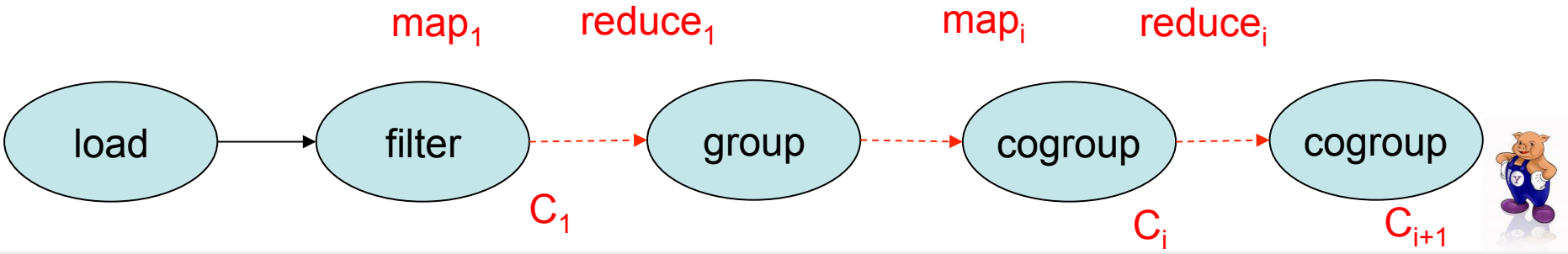


Logical to Physical Plan

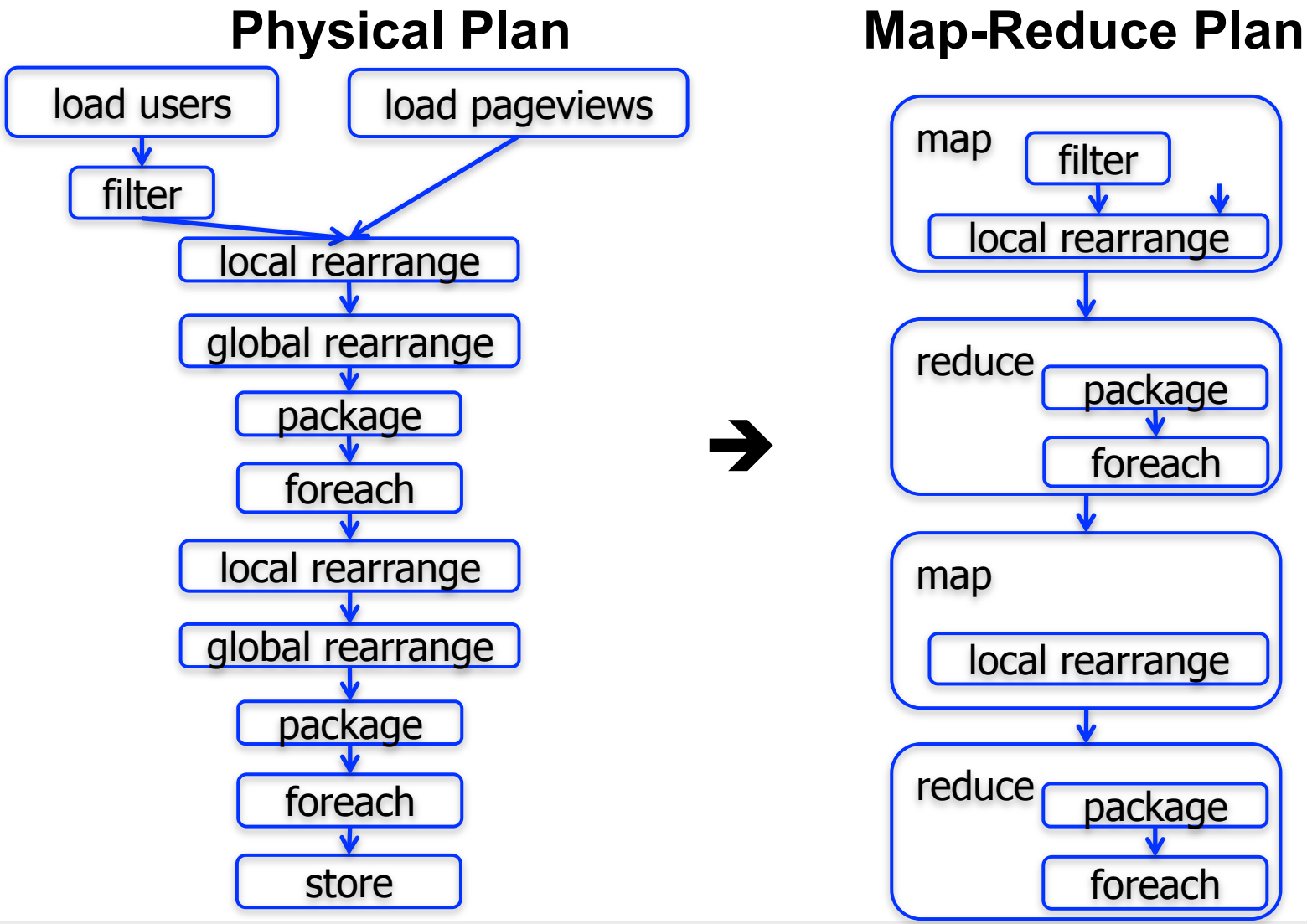


Map Reduce Plan

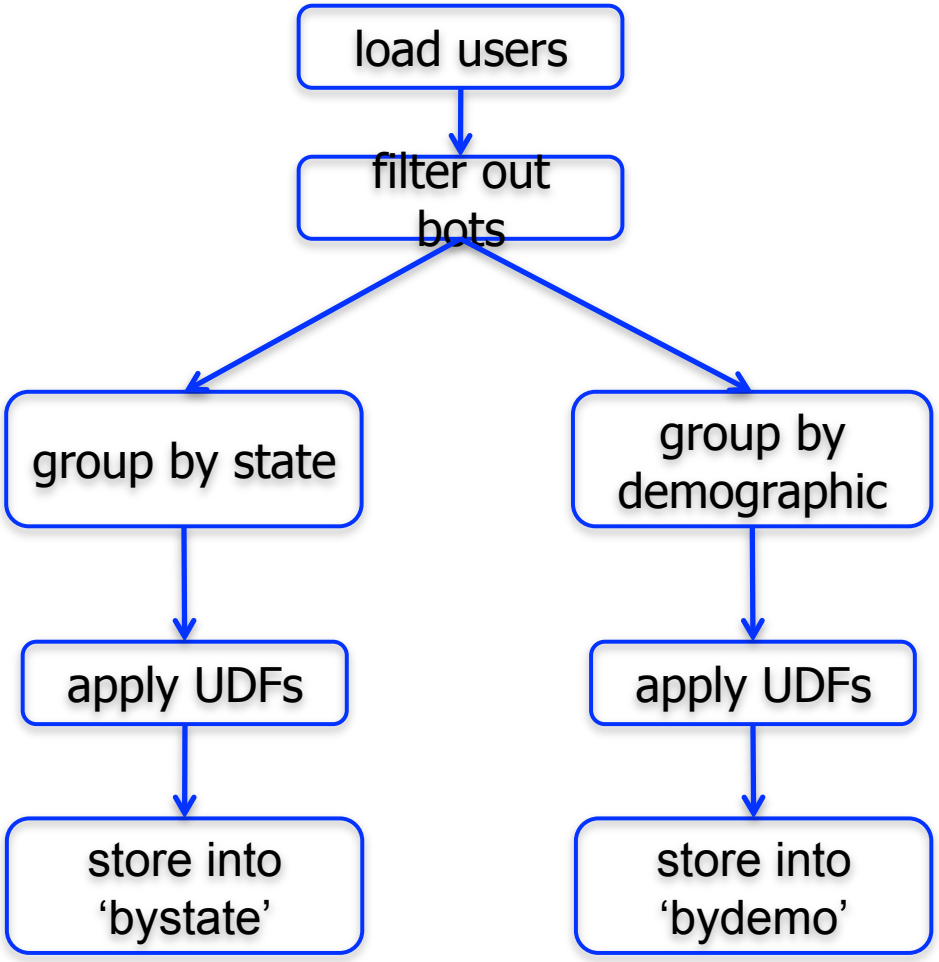
- Physical to Map Reduce (M/R) Plan conversion happens through the MRCompiler
  - Converts a physical plan into a DAG of M/R operators
- Boundaries for M/R include cogroup/group, distinct, cross, order by, limit (in some cases)
  - Push all subsequent operators between cogroup to next cogroup into reduce
  - order by is implemented as 2 M/R jobs
- JobControlCompiler then uses the M/R plan to construct a JobControl object



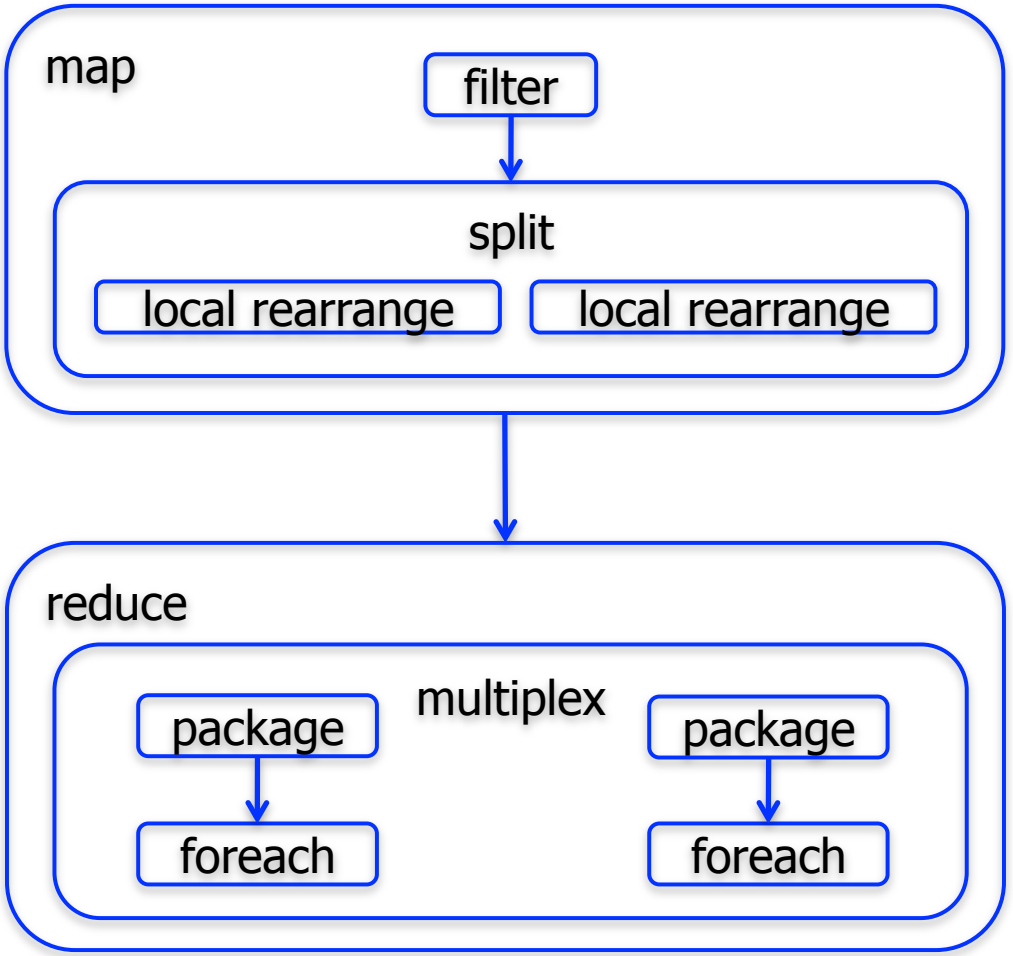
Physical to Map-Reduce Plan



Sharing Scans - Multi query optimization



Multiple Group Map-Reduce Plan



Running Pig

- **Pig can run two run modes or exectypes and they produce the same end results)**
  - **Local mode**:
  - **Hadoop mode** (access to Hadoop cluster and HDFS):
- **Run Pig in 3 ways, in the 2 modes above**
  - **Grunt Shell**: enter Pig commands manually by using Pig's interactive shell, Grunt
  - **Script File**: Place Pig commands in a script file and run the script
  - **Embedded Program**: embed Pig commands in Java and then run the program

Pig Built-in Functions

- Pig has a variety of built-in functions:
  - **Storage**
    - **TextLoader**: for loading unstructured text files. Each line is loaded as a tuple with a single field which is the entire line.
  - **Filter**
    - **isEmpty**: tests if bags are empty
  - **Eval Functions**
    - **COUNT**: computes number of elements in a bag
    - **SUM**: computes the sum of the numeric values in a single-column bag
    - **AVG**: computes the average of the numeric values in a single-column bag
    - **MIN/MAX**: computes the min/max of the numeric values in a single-column bag.
    - **SIZE**: returns size of any datum example map
    - **CONCAT**: concatenate two chararrays or two bytearrays
    - **TOKENIZE**: splits a string and outputs a bag of words
    - **DIFF**: compares the fields of a tuple with arity 2

How to Write a Simple Eval Function

- Eval is the most common function and can be used in FOREACH statement of Pig

```
--myscript.pig
REGISTER myudfs.jar;
A = LOAD 'student_data' AS (name:chararray, age: int,
   gpa: float);
B = FOREACH A GENERATE myudfs.UPPER(name);
DUMP B;
```

Java Source for UPPER UDF

```
package myudfs;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.WrappedIOException;
public class UPPER extends EvalFunc<String>
{
    public String exec(Tuple input) throws IOException
    {
         if (input == null || input.size() == 0)
             return null;
            try
            {
                  String str = (String)input.get(0);
                  return str.toUpperCase();
            }
           catch(Exception e)
           {
              throw WrappedIOException.wrap("Caught exception processing
    input row ", e);
            }
      }
}
```

Eval UDF's specific example

```
package ExpectedClick.Evals;

public class LineAdToMatchtype extends EvalFunc<DataBag>
 {
      private String lineAdSourceToMatchtype (String lineAdSource)
      {
              if (lineAdSource.equals("0")
                  { return "1"; }
              else if (lineAdSource.equals("9")) { return "2"; }
              else if (lineAdSource.equals("13")) { return "3"; }
              else return "0";
       }
  …
```

Eval UDF's example

```
public DataBag exec (Tuple input) throws IOException
{
      if (input == null || input.size() == 0)
            return null;
        String lineAdSource;
        try {
           lineAdSource = (String)input.get(0);
        } catch(Exception e) {
           System.err.println
   ("ExpectedClick.Evals.LineAdToMatchType: Can't convert field to a
   string; error = " + e.getMessage());
           return null;
        }
        Tuple t = DefaultTupleFactory.getInstance().newTuple();
        try {
                    t.set(0,lineAdSourceToMatchtype(lineAdSource));
             }catch(Exception e) {}
         DataBag output = DefaultBagFactory.getInstance
   ().newDefaultBag();
         output.add(t);
        return output;
}
```

Compiling and using Eval UDFs

Create a jar of the UDFs

```
[viraj@machine]$ ls ExpectedClick/Eval
LineAdToMatchtype.java

[viraj@machine]$ javac -cp $PIG_HOME/pig.jar ExpectedClick/Eval/
    *.java

[viraj@machine]$ jar -cf ExpectedClick.jar ExpectedClick/Eval/*
```

Use your function in the Pig Script

```
register ExpectedClick.jar;
offer = LOAD '/user/viraj/dataset' USING PigStorage() AS (a,b,c);

convertedoffer = FOREACH offer GENERATE a AS query,
FLATTEN(ExpectedClick.Evals.LineAdToMatchtype((chararray)b)) AS
    matchtype, ..
```

Aggregate Functions

- Aggregate functions are another type of eval function usually applied to grouped data
- Takes a bag and returns a scalar value
- Aggregate functions can use the Algebraic interface to perform intermediate computations in the Combiner

```
A = LOAD 'student_data' AS (name: chararray, age:
   int, gpa: float);
B = GROUP A BY name;
C = FOREACH B GENERATE group, COUNT(A);
DUMP C;
```

Algebriac Interface

- Algebriac Inteface consists of the following functions:
  - getInitial()
    - exec function of the Initial class is called once and is passed the original input tuple and is called in the Map
  - getIntermed()
    - exec function of the Intermed class can be called zero or more times by the Combiner
  - getFinal()
    - exec function of the Final class is invoked once by the Reducer

COUNT Aggregate function

```
public class COUNT extends EvalFunc<Long> implements Algebraic
{
   public Long exec(Tuple input) throws IOException {
     return count(input);
   }
   public String getInitial() {
     return Initial.class.getName();
   }
   public String getIntermed() {
    return Intermed.class.getName();
   }
   public String getFinal() {
     return Final.class.getName();
   }
  static public class Initial extends EvalFunc<Tuple>
  {
      public Tuple exec(Tuple input) throws IOException {
       return TupleFactory.getInstance().newTuple(count(input));
     }
   }
  static public class Intermed extends EvalFunc<Tuple>
  {
      public Tuple exec(Tuple input) throws IOException {
          return TupleFactory.getInstance().newTuple(sum(input));
  }
}
```

COUNT Aggregate function

```
static public class Final extends EvalFunc<Long>
 {
     public Tuple exec(Tuple input) throws IOException {return sum(input);}
 }
 static protected Long count(Tuple input) throws ExecException {
       Object values = input.get(0);
       if (values instanceof DataBag)
           return ((DataBag)values).size();
       else if (values instanceof Map)
       return new Long(((Map)values).size());
 }

 static protected Long sum(Tuple input) throws ExecException, NumberFormatException {
     DataBag values = (DataBag)input.get(0);
     long sum = 0;
     for (Iterator (Tuple) it = values.iterator(); it.hasNext();) {
         Tuple t = it.next();
         sum += (Long)t.get(0);
     }
     return sum;
   }
}
```

Filter Function

- Filter functions are eval functions that return a boolean value
- Filter functions can be used anywhere a Boolean expression is appropriate
  - FILTER operator or Bincond
- Example use Filter Func to implement outer join

```
A = LOAD 'student_data' AS (name: chararray, age: int, gpa:
   float);
B = LOAD 'voter_data' AS (name: chararray, age: int,
   registration: chararay, contributions: float);
C = COGROUP A BY name, B BY name;
D = FOREACH C GENERATE group, flatten((IsEmpty(A) ? null : A)),
   flatten((IsEmpty(B) ? null : B));
dump D;
```

isEmpty FilterFunc

```
import java.io.IOException;
import java.util.Map;
import org.apache.pig.FilterFunc;
import org.apache.pig.backend.executionengine.ExecException;
import org.apache.pig.data.DataBag;
import org.apache.pig.data.Tuple;
import org.apache.pig.data.DataType;
import org.apache.pig.impl.util.WrappedIOException;
public class IsEmpty extends FilterFunc
{
      public Boolean exec(Tuple input) throws IOException
      {
          if (input == null || input.size() == 0) return null;
          try {
                    Object values = input.get(0);
                    if (values instanceof DataBag)
                        return ((DataBag)values).size() == 0;
                    else if (values instanceof Map)
                        return ((Map)values).size() == 0;
                    else {
                        throw new IOException("Cannot test a " + DataType.findTypeName(values) + "
    for emptiness.");
                    }
                }
            catch (ExecException ee) {
              throw WrappedIOException.wrap("Caught exception processing input row ", ee);
            }
       }
}
```

Schema specifications in UDF

- The below script does not work correctly

```
register myudfs.jar;
A = load 'student_data' as (name:chararray, age:int, gpa:float);
B = foreach A generate flatten(myudfs.Swap(name, age)), gpa;
C = foreach B generate $2;
D = limit B 20;
dump D;
Error java.io.IOException: Out of bound access. Trying to access non
   existent column: 2. Schema {bytearray,gpa: float} has 2 column(s).
```

- If a UDF returns a tuple or a bag and schema information is not provided
  - Pig assumes that the tuple contains a single field of type bytearray

Swap UDF

```
package myudfs;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.data.TupleFactory;
import org.apache.pig.impl.logicalLayer.schema.Schema;
import org.apache.pig.data.DataType;

public class Swap extends EvalFunc<Tuple> {
    public Tuple exec(Tuple input) throws IOException {
        if (input == null || input.size()   2
            return null;
        try{
            Tuple output = TupleFactory.getInstance().newTuple(2);
            output.set(0, input.get(1));
            output.set(1, input.get(0));
            return output;
        } catch(Exception e){
            System.err.println("Failed to process input; error - " + e.getMessage());
            return null;
        }
}
```

Swap UDF now containing Schema info

```
public Schema outputSchema(Schema input) {
        try{
            Schema tupleSchema = new Schema();
            tupleSchema.add(input.getField(1));
            tupleSchema.add(input.getField(0));
            return new Schema(new Schema.FieldSchema(getSchemaName(this.getClass
    ().getName().toLowerCase(), input),tupleSchema, DataType.TUPLE));
        }catch (Exception e){
                return null;
        }
    }
}
```

B = foreach A generate flatten(myudfs.Swap(name, age)), gpa;
describe B;

B: {myudfs.swap_age_3::age: int,myudfs.swap_age_3::name:chararray,gpa: float}

Accumulator Interface

- Normally Pig passes the entire bag from a group/cogroup to UDF's
- Using the Accumulator interface, Pig guarantees that the data for the same key is passed continuously but in small increments

```
public interface Accumulator <T> {
   /**
    * Process tuples. Each DataBag may contain 0 to many tuples for current key
    */
    public void accumulate(Tuple b) throws IOException;
    /**
     * Called when all tuples from current key have been passed to the accumulator.
     * @return the value for the UDF for this key.
     */
    public T getValue();
    /**
     * Called after getValue() to prepare processing for next key.
     */
    public void cleanup();
}
```

IntMax UDF using Accumulator Interface

```
public class IntMax extends EvalFunc<Integer> implements Algebraic,
    Accumulator<Integer>
{
    /* Accumulator interface */
    private Integer intermediateMax = null;
    @Override
    public void accumulate(Tuple b) throws IOException {
        try {
            Integer curMax = max(b);
            if (curMax == null) {
                return;
            }
            /* if bag is not null, initialize intermediateMax to negative infinity */
            if (intermediateMax == null) {
                intermediateMax = Integer.MIN_VALUE;
            }
            intermediateMax = java.lang.Math.max(intermediateMax, curMax);
        } catch (ExecException ee) {
            throw ee;
        } catch (Exception e) {
            int errCode = 2106;
            String msg = "Error while computing max in " + this.getClass
    ().getSimpleName();
            throw new ExecException(msg, errCode, PigException.BUG, e);
        }
}
```

IntMax using Accumulator Interface

```
 @Override
 public void cleanup() {
      intermediateMax = null;
  }
  @Override
    public Integer getValue() {
        return intermediateMax;
    }
}
```

Load Function

- LoadFunc abstract class has the main methods for loading data
- 3 important interfaces
  - LoadMetadata has methods to deal with metadata
  - LoadPushDown has methods to push operations from pig runtime into loader implementations
  - LoadCaster has methods to convert byte arrays to specific types
    - implement this method if your loader casts (implicit or explicit) from DataByteArray fields to other types
- Functions to be implemented
  - getInputFormat()
  - setLocation()
  - prepareToRead()
  - getNext()
  - *setUdfContextSignature()*
  - *relativeToAbsolutePath()*

Regexp Loader Example

```
public class RegexLoader extends LoadFunc {
    private LineRecordReader in = null;
    long end = Long.MAX_VALUE;
    private final Pattern pattern;
        public RegexLoader(String regex) {
                pattern = Pattern.compile(regex);
        }
     public InputFormat getInputFormat() throws IOException {
            return new TextInputFormat();
        }
      public void prepareToRead(RecordReader reader, PigSplit split)
          throws IOException {
            in = (LineRecordReader) reader;
        }
        public void setLocation(String location, Job job) throws IOException {
            FileInputFormat.setInputPaths(job, location);
        }
```

Regexp Loader

```
public Tuple getNext() throws IOException {
        if (!in.nextKeyValue()) {
            return null;
        }
        Matcher matcher = pattern.matcher("");
        TupleFactory mTupleFactory = DefaultTupleFactory.getInstance();
        String line;
        boolean tryNext = true;
        while (tryNext) {
            Text val = in.getCurrentValue();
            if (val == null) {
                break;
            }
            line = val.toString();
            if (line.length() > 0 && line.charAt(line.length() - 1) == '\r') {
                line = line.substring(0, line.length() - 1);
            }
            matcher = matcher.reset(line);
ArrayList<DataByteArray> list = new ArrayList<DataByteArray>();
            if (matcher.find()) {
                tryNext=false;
                for (int i = 1; i <= matcher.groupCount(); i++) {
                    list.add(new DataByteArray(matcher.group(i)));
                }
                return mTupleFactory.newTuple(list);
            }
        }

        return null;
    } }
```

Embed Pig Latin in Java

```
/* create a pig server in the main class*/
{
  PigServer pigserver = new PigServer(args[0]);
   runMyQuery(pigServer, "/user/viraj/mydata.txt")
}

/* submit in function runMyQuery */

runMyQuery(PigServer pigServer, String inputFile) throws
IOException {
       pigServer.registerQuery("A = load '" + inputFile +
"' as (f1,f2,f3);");
       pigServer.registerQuery("B = group A by f1;");
       pigServer.registerQuery("C = foreach B generate
flatten(group);");
       pigServer.store("C", "/user/viraj/myoutput");
}
```

Use EXPLAIN to Understand Logical, Physical & M/R Plan

```
grunt>sportsviews = load 'sportsviews.txt' as (userId: chararray,team:chararray,timestamp: int);
grunt>groupsportsviews = group sportsviews by userId;
grunt> describe group_sportsviews;
groupsportsviews: {group: chararray,sports_views: {userId: chararray,team: chararray,timestamp:
    integer}}
grunt> dump sportsviews;
(alice,{(alice,lakers,3),(alice,lakers,7)})
grunt> explain groupsportsviews
```

```
-----------------------------------------------
| Map Reduce Plan                             |
-----------------------------------------------
MapReduce node viraj-Sat Oct 25 20:38:23 UTC 2008-2261
Map Plan
Local Rearrange[tuple]{chararray}(false) - viraj-Sat Oct 25 20:38:23 UTC 2008-2258
|   |
|   Project[chararray][0] - viraj-Sat Oct 25 20:38:23 UTC 2008-2259
|
|---New For Each(false,false,false)[bag] - viraj-Sat Oct 25 20:38:23 UTC 2008-2255
    |   |
    |   Cast[chararray] - viraj-Sat Oct 25 20:38:23 UTC 2008-2250
    |   |
    |   |---Project[bytearray][0] - viraj-Sat Oct 25 20:38:23 UTC 2008-2249
    |   |
    |   Cast[chararray] - viraj-Sat Oct 25 20:38:23 UTC 2008-2252
    |   |
    |   |---Project[bytearray][1] - viraj-Sat Oct 25 20:38:23 UTC 2008-2251
    |   |
    |   Cast[integer] - viraj-Sat Oct 25 20:38:23 UTC 2008-2254
    |   |
    |   |---Project[bytearray][2] - viraj-Sat Oct 25 20:38:23 UTC 2008-2253
    |
    |---Load(sports_views.txt:org.apache.pig.builtin.PigStorage) - viraj-Sat Oct 25 20:38:23 UTC 2008-2248--------
Reduce Plan
Store(fakefile:org.apache.pig.builtin.PigStorage) - viraj-Sat Oct 25 20:38:23 UTC 2008-2260
|
|---Package[tuple]{chararray} - viraj-Sat Oct 25 20:38:23 UTC 2008-2257--------
Global sort: false
----------------
```

LIMIT Reduces Records for Debugging

- **`LIMIT`** allows to limit the number of output tuples produced
- Where possible limit is pushed up the execution pipeline to drop records as soon as possible (But no guarantee on which rows are returned)
- No order guarantees, except when **`LIMIT`** *immediately* follows **`ORDER BY`**

```
grunt> sports_views = load 'sports_views_long.txt' as
(userId: chararray,team: chararray,timestamp : int);
grunt> sport_vieworder = order sports_views by timestamp;
grunt> sports_viewlimit = limit sport_vieworder 10;
grunt> dump sports_viewlimit
```

Pig LIMIT Example

```
[viraj@gsgw1011 ~/pigscripts]$ pig --latest --x local
grunt> sports_views = load 'sports_views_long.txt' as (userId: chararray,team: chararray,timestamp : int);
grunt> sports_viewslimit = limit sports_views 10;
grunt> dump sports_views;
(alice,lakers,7)
(alan,sun,5)
(peter,knicks,12)
(pan,suns,1)
(mary,timberwolves,2)
(bill,nets,2)
(john,warriors,4)
(jason,heat,56)
(alice,lakers,7)
(alice,lakers,7)
(alice,lakers,7)
(alice,lakers,7)
(alice,lakers,7)
(alan,sun,5)
(peter,knicks,12)
(pan,suns,1)
(mary,timberwolves,2)
(bill,nets.2)
```

```
grunt> dump sports_viewslimit;
(pan,suns,1)
(alan,sun,5)
(bill,nets,2)
(john,warriors,4)
(mary,timberwolves,2)
(alice,lakers,7)
(alice,lakers,7)
(alice,lakers,7)
(jason,heat,56)
(peter,knicks,12)
```

How to Increase Performance of Pig Scripts

- **Project Early project and Often**
  - Pig does not (yet) determine when a field is no longer needed or drop the field from the row
  - Performance improvement of 50% in some cases
- **Filter Early and Often**
- **Drop Nulls Before a Join**
  - Performance improvement of 10x when 7% of keys were null
- **Prefer `DISTINCT` over `GROUP BY - GENERATE`**
  - For extracting the unique values from a column in a relation
  - 20x faster in some cases
- **Use the right type of data** whenever possible
  - Results in better parse time error-checking
  - Efficient execution

How to Increase Performance of Pig Scripts

Not Optimized

```
A = load 'myfile' as (t, u, v);
B = load 'myotherfile' as (x, y, z);
C = join A by t, B by x;
D = group C by u;
E = foreach D generate group, COUNT($1);
```

Optimized

```
A = load 'myfile' as (t, u, v);
A1 = foreach A generate t, u;
B = load 'myotherfile' as (x, y, z);
B1 = foreach B generate x;
C = join A1 by t, B1 by x;
C1 = foreach C generate t, u;
D = group C1 by u;
E = foreach D generate group, COUNT($1);
```

project early

```
A = load 'myfile' as (t, u, v);
B = load 'myotherfile' as (x, y, z);
C = join A by t, B by x;
```

```
A = load 'myfile' as (t, u, v);
B = load 'myotherfile' as (x, y, z);
C1 = cogroup A by t, B by x;
C = foreach C1 generate flatten(A), flatten(B);
```

```
A = load 'myfile' as (t, u, v);
B = load 'myotherfile' as (x, y, z);
A1 = filter A by t is not null;
B1 = filter B by x is not null;
C = join A1 by t, B1 by x;
```

drop nulls before join

```
A = load 'myfile' as (t, u, v);
B = foreach A generate u;
C = group B by u;
D = foreach C generate group as uniquekey;
dump D;
```

```
A = load 'myfile' as (t, u, v);
B = foreach A generate u;
C = distinct B;
dump C;
```

prefer distinct

How to Increase Performance of Pig Scripts

Not Optimized

```
A = load 'myfile' as (t, u, v);
B = load 'myotherfile' as (x, y, z);
C = filter A by t == 1;
D = join C by t, B by x;
E = group D by u;
F = foreach E generate group, COUNT($1);
```

Optimized

```
A = load 'myfile' as (t, u, v);
B = load 'myotherfile' as (x, y, z);
C = join A by t, B by x;
D = group C by u;
E = foreach D generate group, COUNT($1);
F = filter E by C.t == 1;
```

filter early and often

```
A = load 'myfile' as (t, u, v);
B = foreach A generate t + u;
```

```
A = load 'myfile' as (t: int, u: int, v);
B = foreach A generate t + u;
```

use types

```
A = load 'data' as (in: map[]);
-- get key out of the map
B = foreach A generate in#k1 as k1, in#k2 as k2;
-- concatenate the keys
C = foreach B generate CONCAT(k1, k2);
.......
```

```
A = load 'data' as (in: map[]);
-- concatenate the keys from the map
B = foreach A generate CONCAT(in#k1, in#k2);
....
```

reduce operator pipeline

Increase Number of Reducers

- **Map parallelism depends on data size**
  - Approximately one map per 128MB data on the Grid
  - Number of maps created for your M/R depends on the splits produced by input format (and presently not configurable in Pig)

- **Reduce parallelism can be defined with PARALLEL keyword**
  - Can be put on any **GROUP**, **COGROUP** or **JOIN** construct (but doesn't affect **FOREACH** - map only)

    J = JOIN A by url, B by url PARALLEL 20

    (20 reducers will be used in your M/R job)
  - If **PARALLEL** is not specified then 1 reducer is used
  - "**set default_parallel constant**" in Pig script will set same number of reducers for every M/R job

Increase Performance of Pig Scripts

- **When doing a join/cogroup, place your largest data or table last**

```
small = load 'data1';
large = load 'data2';
myoptjoin = join small by $0, large by $0;
```

- **Write queries to invoke combiner as 10x performance improvements are observed when using it**

Use of Algebraic interface when writing UDFs:

```
Y = group X by $0;
Z = foreach Y generate group, COUNT(X), SUM(X);

Y = group X by $0, $1;
Z = foreach Y generate flatten(group), SUM(X);

Y = group X all;
Z = foreach Y generate  COUNT(X);

Y = group X by $0;
Z = foreach Y generate COUNT(X), group;
```

Fragment Replicate Join

- **Use Fragment Replicate Join**
  - If you have a small (< 100M) table, then

```
J = join big by $0, small by $0 using "replicated";
```

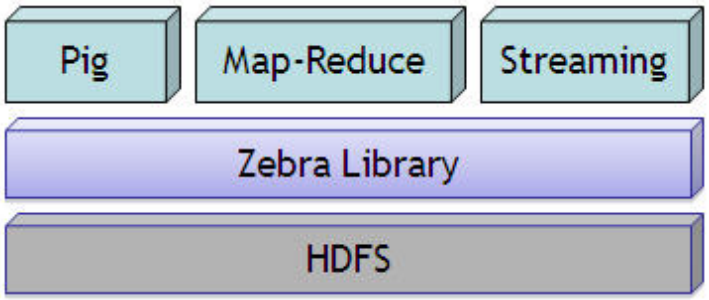
- **Distribute processing of huge files** by **fragmenting** it – then, **replicate** the small file to all machines (which have a fragment of the huge file)
  - Basically written as Pig UDF
  - http://wiki.apache.org/pig/PigFRJoin

Use Skewed Join

- Parallel joins are vulnerable to the presence of skew in the underlying data
- If the underlying data is sufficiently skewed, load imbalances will swamp any of the parallelism gains
- Skewed join can be used when the underlying data is sufficiently skewed and you need a finer control over the allocation of reducers to counteract the skew

```
big = LOAD 'big_data' AS (b1,b2,b3);
massive = LOAD 'massive_data' AS (m1,m2,m3);
C = JOIN big BY b1, massive BY m1 USING "skewed";
```

Zebra



- Zebra is an access path library for reading and writing data in a column-oriented fashion
- Zebra functions as an abstraction layer between your client application and data on the Hadoop Distributed File System (HDFS)
- Zebra supports client applications written as Pig, MapReduce, or Streaming

Zebra and Pig

- Loading Data

```
register /grid/0/gs/pig/current/libexec/released/zebra.jar;
A = LOAD 'studenttab' USING org.apache.hadoop.zebra.pig.TableLoader();
B = FOREACH A GENERATE name, age, gpa;
```

- Map Side and Merge Join

```
A = LOAD'studenttab' USING org.apache.hadoop.zebra.pig.TableLoader('',
    'sorted');
B = LOAD 'votertab' USING org.apache.hadoop.zebra.pig.TableLoader('',
    'sorted');
G = JOIN A BY $0, B By $0 USING "merge";
```

  - Map-side group
  - Loader will perform sort-preserving merge to make sure that the data is globally sorted

```
A = LOAD 'studentsortedtab, studentnullsortedtab' using
    org.apache.hadoop.zebra.pig.TableLoader('name, age, gpa, source_table',
    'sorted');
B = GROUP A BY $0 USING "collected";
C = FOREACH B GENERATE group, MAX(a.$1);
```

Pig Resources

- **Documentation**
  - General info: http://wiki.apache.org/pig/
  - Pig Documentation + UDF: http://hadoop.apache.org/pig/docs/r0.7.0/
- **Mailing lists**
  - External: pig-user@hadoop.apache.org
- **Issue-tracking**
  - External: http://issues.apache.org/jira/browse/PIG