Pig Latin

User Defined Functions
Wrapping
Examples

User Defined functions (UDF)

UDFs

- Apache Pig provides an extensive support for **U**ser **D**efined **F**unctions
- This gives the programmers the ability to define their own function and use them in the Pig Latin script
- These functions can be written in any of the following languages
  - Java, Python, Ruby, JavaScript, Groovy, Jython

Write UDFs in Java

- you need to add the pig-x-x-x.jar file in your project
  - if your are managing your project using maven, then you can add dependencies in your pom.xml file

```
<dependency>
        <groupId>org.apache.pig</
groupId>
        <artifactId>pig</
artifactId>
        <version>0.15.0</version>
</dependency>
```

Create a Java class

```
import java.io.IOException;

import org.apache.pig.EvalFunc;

import org.apache.pig.data.Tuple;



public class Eval_Upper extends EvalFunc<String>{


   public String exec(Tuple input) throws IOException {

      if (input == null || input.size() == 0)

      return null;

      String str = (String)input.get(0);

      return str.toUpperCase();

   }

}
```

Code

- The UDF class must
  - inherit the EvalFunc from the pig library
    - imported: import org.apache.pig.EvalFunc;
  - implement exec() method
- After finishing the code, create a jar file

Use the UDF

- First, we need to register the jar file containing the new function
    - > REGISTER ‘jarFile’ ;
- Second, we need to give it an alias (name)
    - > DEFINE **Eval_Upper** Eval_Upper() ;
- Now, we can use it by its name in the Pig Latin code

```
grunt> student_details = LOAD 'student_details.txt' USING PigStorage(‘,') as (id:int, name:chararray, city:chararray);
grunt> student_upper = foreach student_details  Generate Eval_Upper (name);
```

Pig Latin

Wrapping

Examples

Word Count Example
Pig Latin

- Input files contain lines of text
- The output should contains two fields; word & frequency

Pig Latin code

Pig Latin is a data flow language
Pig runs on top of Hadoop
……

1. Load the input files
    1. input_lines = load ‘files.txt’ AS (line:chararray);
2. For each line tokenize it and generate rows; each row represent one word in the line

words = FOREACH input GENERATE Flatten(TOKENIZE(line)) AS word;

TOKENIZE(line) produces bag of words first line: {(pig), (latin), (is), (a), (data), (flow), (language)}

Flatten will covert columns into rows

(pig)
(latin)
(is)
(a)
(data)
(flow)
(language)

Pig Latin code

words

(pig)
(latin)
(is)
(a)
(data)
(flow)
(language)
(pig)
(runs)
(on)
(top)
(of)
(Hadoop)

3. Group by word

   group_word = group words BY word ;

4. Count

   word_count = FOREACH group_word GENERATE group, COUNT(words)

5. store result

   1. STORE word_count INTO ‘’ ;

Data Analysis Task

- Top 10 most visited pages for each category

visits

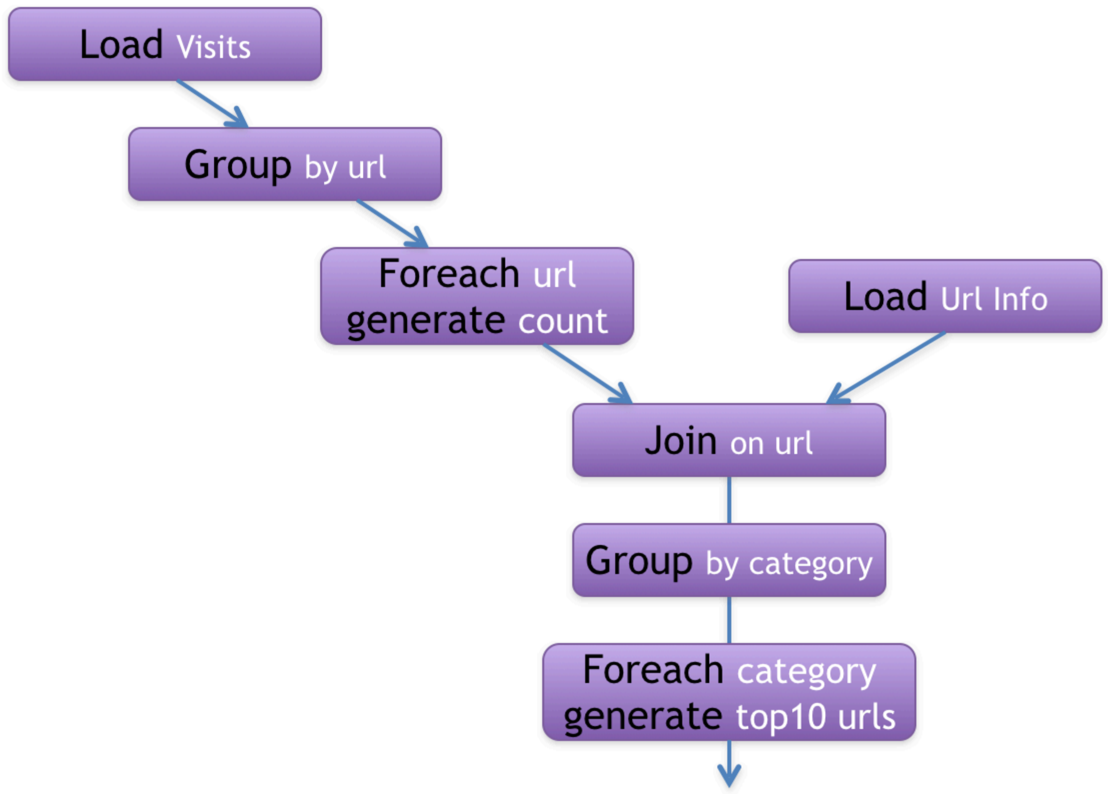
User	Url	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

Url Info

Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

- Draw the flow of how to achieve the task
- Write Pig Latin code (assume that the datasets above are comma separated text file)

Flow

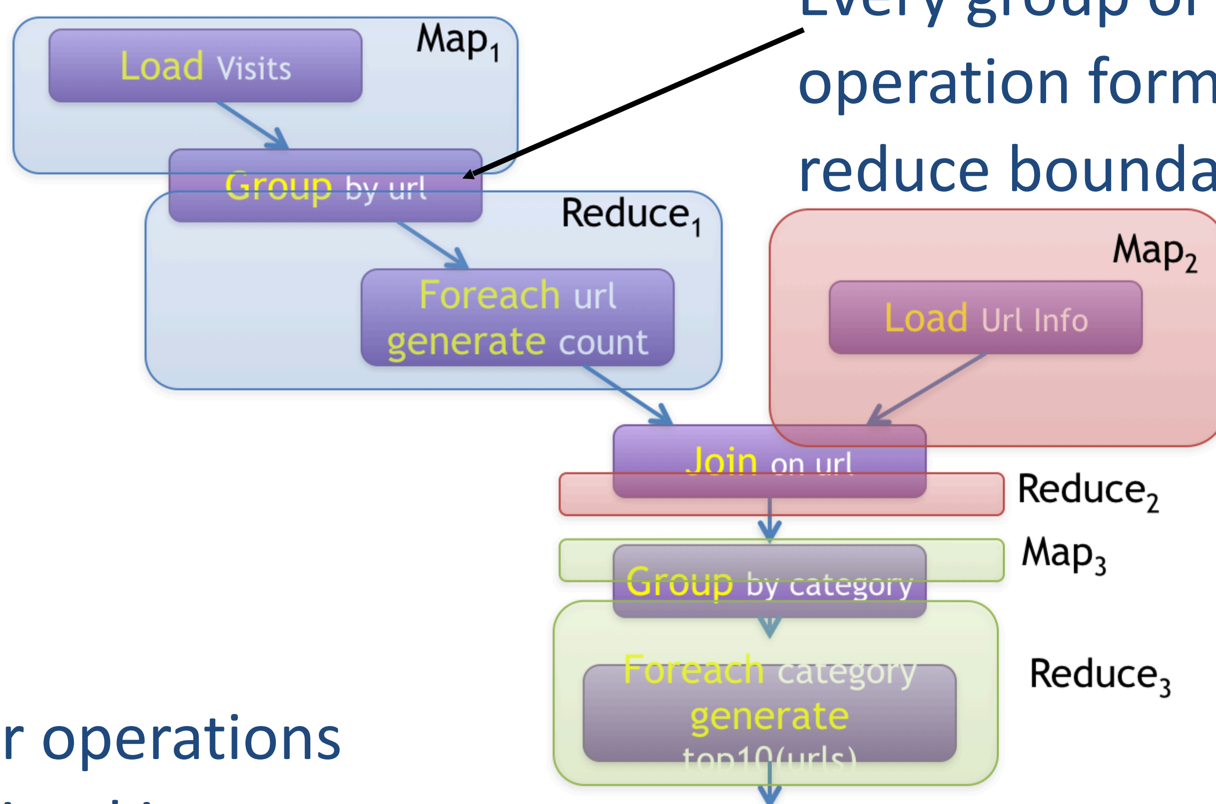

Load Visits
Group by url
Foreach url
generate count
Load Url Info
Join on url
Group by category
Foreach category
generate top10 urls

Pig Latin code

```
visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts  = foreach gVisits generate url, count(visits);

urlInfo = load '/data/urlInfo' as (url, category, pRank);

visitCounts  = join visitCounts by url, urlInfo by url;

gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

Compilation Into MapReduce

Every group or join operation forms a map-reduce boundary



Other operations pipelined into map and reduce phases

Data Flow Language

- Step-by-step procedural Language
- Users specify a sequence of steps where each step represents a single high-level data transformation
- Compared to a SQL, it is easier to keep track of the variable and where are you in the process

Pig Latin vs. SQL

- (url, category, pagerank) dataset,
- query that finds,
  - *For each sufficiently large category (*> 10^6*), the average pagerank of high-pagerank urls (pagerank* > *0.2) in that category*
- Write the SQL query to achieve the above task
- Then write the Pig Latin code

Pig Latin vs. SQL

- (url, category, pagerank) dataset,
- query that finds,
  - *For each sufficiently large category (> 10^6), the average pagerank of high-pagerank urls (pagerank > 0.2) in that category*

```
SELECT category, Avg(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 10^6
```

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY       COUNT(good_urls) > 10^6 ;
output = FOREACH big_groups GENERATE                  category, AVG(good_urls.pagerank);
```

Schema is optional
can be assigned dynamically

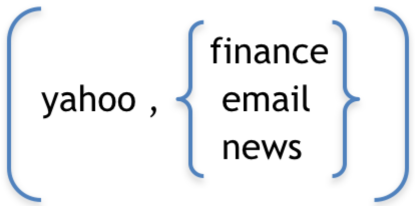
```
visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts  = foreach gVisits generate url, count(visits);

urlInfo = load '/data/urlInfo' as (url, category, pRank);

visitCounts  = join visitCounts by url, urlInfo by url;

gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

User Defined function are very supported

```
visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts  = foreach gVisits generate url, count(visits);

urlInfo = load '/data/urlInfo' as (url, category, pRank);

visitCounts  = join visitCounts by url, urlInfo by url;

gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

Nested Data Model

- Atomic values, tuples, bags, map
- helpful
  - avoid having expensive joins



Testing data sets

http://www.gutenberg.org/cache/epub/100/pg100.txt

http://www.gutenberg.org/cache/epub/31100/pg31100.txt

http://www.gutenberg.org/cache/epub/3200/pg3200.txt

http://www.gutenberg.org/cache/epub/2253/pg2253.txt

http://www.gutenberg.org/cache/epub/1513/pg1513.txt

http://www.gutenberg.org/cache/epub/1120/pg1120.txt