Introduction to Database Systems
CSE 444

Lecture 23: Pig Latin (continued)
Review: Example from Lecture 22

• Input: a table of urls:
  (url, category, pagerank)

• Compute the average pagerank of all sufficiently high pageranks, for each category

• Return the answers only for categories with sufficiently many such pages
First in SQL…

```
SELECT category, AVG(pagerank)
FROM urls
WHERE pagerank > 0.2
GROUP By category
HAVING COUNT(*) > 10^6
```
Pig Latin combines
• high-level declarative querying in the spirit of SQL, and
• low-level, procedural programming a la map-reduce.
We Also Saw JOIN Last Time

results:       {((queryString, url, position))}
revenue:      {((queryString, adSlot, amount))}

join_result = JOIN results BY queryString
              revenue BY queryString

join_result : {((queryString, url, position, adSlot, amount))}
Today: Cogroup

• A generic way to group tuples from two datasets together
Co-Group

Dataset 1 results: 
{(queryString, url, position)}

Dataset 2 revenue: 
{(queryString, adSlot, amount)}

```
grouped_data =
    COGROUP results BY queryString,
    revenue BY queryString;

    grouped_data: 
    {(queryString, 
        results:{(url, position)},
        revenue:{(adSlot, amount)})}
```

What is the output type in general?

```
{group_id, bag dataset 1, bag dataset 2}
```
Co-Group

Is this an inner join or an outer join?

CSE 444 - Spring 2009
Co-Group

grouped_data: {(queryString, results:{(url, position)}, revenue:{(adSlot, amount)})}

url_revenues = FOREACH grouped_data
         GENERATE
         FLATTEN(distributeRevenue(results, revenue));

distributeRevenue is a UDF that accepts search results and revenue information for a query string at a time, and outputs a bag of urls and the revenue attributed to them.
Co-Group v.s. Join

grouped_data: {(queryString, results:{(url, position)},
                   revenue:{(adSlot, amount)})}

result is the same as JOIN
Asking for Output: STORE

```
STORE query_revenues INTO `myoutput`
USING myStore();
```

Meaning: write `query_revenues` to the file `myoutput`

This is when the entire query is finally executed!
Query Processing Steps

Pig Latin program

Pig parser

Pig compiler

Pig MR compiler

Execution plan

MR jobs

Hadoop

Disk A

Disk B

Load

Filter

Join

Map 1

Reduce 1

Map 2

Disk

Disk

Disk
Implementation

• Over Hadoop!
• Parse query:
  – All between LOAD and STORE \(\rightarrow\) one logical plan
• Logical plan \(\rightarrow\) ensemble of MapReduce jobs
  – Each (CO)Group becomes a MapReduce job
  – Other ops merged into Map or Reduce operators
• Extra MapReduce jobs for sampling before SORT operations
Implementation
Advice for the Project

• Always run first locally
  – Test your program on your local machine, on a smaller dataset
  – After you debugged the program, send it to the cluster

• Batch processing:
  – Keep in mind that Hadoop does batch processing
  – Your job takes 2-7 minutes on the cluster
  – No-one else can run on the same compute nodes during this time !!
Longer Example: Tutorial Script 2

- Goal: Process a search query log file and find search phrases that occur with particular high frequency during certain times of the day

```
raw = LOAD 'excite-small.log'
    USING PigStorage("	") AS (user, time, query);

clean1 = FILTER raw BY
    org.apache.pig.tutorial.NonURLDetector(query);
```
Longer Example: Tutorial Script 2

clean2 = FOREACH clean1
    GENERATE user, time,
    org.apache.pig.tutorial.ToLower(query)
    as query;

houred = FOREACH clean2
    GENERATE user,
    org.apache.pig.tutorial.ExtractHour(time)
    AS hour, query;
Longer Example: Tutorial Script 2

\[
ngramed1 = \text{FOREACH} \ houred \\
\quad \quad \quad \text{GENERATE} \ \text{user, hour,}
\quad \quad \quad \quad \text{flatten(}\text{org.apache.pig.tutorial.NGramGenerator(query)}\text{)}
\quad \quad \quad \text{AS} \ ngram;
\]

\[
ngramed2 = \text{DISTINCT} \ ngramed1;
\]

\[
hour\_frequency1 = \text{GROUP} \ ngramed2 \ \text{BY} \ (ngram, \ \text{hour});
\text{hour\_frequency2} = \text{FOREACH} \ \text{hour\_frequency1} \\
\quad \text{GENERATE} \ \text{flatten}($0), \ \text{COUNT}($1) \ \text{as} \ \text{count};
\]
Longer Example: Tutorial Script 2

\[\text{hour\_frequency3} = \text{FOREACH hour\_frequency2 GENERATE }\]$0\ as\ ngram,\ $1\ as\ hour,\ $2\ as\ count;\]

\[\text{hour00} = \text{FILTER hour\_frequency2 BY hour eq '00';}\]
\[\text{hour12} = \text{FILTER hour\_frequency3 BY hour eq '12';}\]

\[\text{same} = \text{JOIN hour00 BY }$0,\ \text{hour12 BY }$0;\]
Longer Example: Tutorial Script 2

same1 = FOREACH same
    GENERATE hour_frequency2::hour00::group::ngram
    AS ngram, $2 as count00, $5 as count12;

STORE same1
    INTO 'script2-local-results.txt' USING PigStorage();