# Introduction to Data Management
# CSE 344

## Lecture 27: Map Reduce and Pig Latin

# Announcements

- HW8 out now, due last Thursday of the qtr
  - You should have received AWS credit code via email. Send mail to cse344-staff@cs if problems.
  - Start setting up account now(!). Takes time.
    - And **follow instructions!!** Usually the biggest problem.
- Final exam:
  - Mon. 12/8, 2:30-4:20, this room
  - Review Sun. 12/7, 2 pm, EE 037
  - Comprehensive (but no rel. calculus, datalog)
  - Closed book; reference notes included in test
- HW grading: yes, we're behind. Should have hw4 and hw5 by Friday night. HW5 and HW6 solutions posted now.

# Outline

- Example of a large MapReduce job
- Whirlwind tour of Pig Latin for HW8
  - You'll need to learn from slides, starter code, Hadoop and related web pages; will not do details in class like we did for SQL

# Executing a Large MapReduce Job

# Anatomy of a Query Execution

- Running problem #4
- 20 nodes = 1 master + 19 workers
- Using PARALLEL 50

# March 2013



## Hadoop job_201303091944_0001 on domU-12-31-39-06-75-A1

**User:** hadoop
**Job Name:** PigLatin:DefaultJobName
**Job File:** hdfs://10.208.122.79:9000/mnt/var/lib/hadoop/tmp/mapred/staging/hadoop/.staging/job_201303091944_0001/job.xml
**Submit Host:** domU-12-31-39-06-75-A1.compute-1.internal
**Submit Host Address:** 10.208.122.79
**Job-ACLs: All users are allowed**
**Job Setup:** Successful
**Status:** Succeeded
**Started at:** Sat Mar 09 19:49:21 UTC 2013
**Finished at:** Sat Mar 09 23:33:14 UTC 2013
**Finished in:** 3hrs, 43mins, 52sec
**Job Cleanup:** Successful
**Black-listed TaskTrackers:** 1

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 7908 | 0 | 0 | 7908 | 0 | 14 / 16 |
| reduce | 100.00% | 50 | 0 | 0 | 50 | 0 | 0 / 8 |

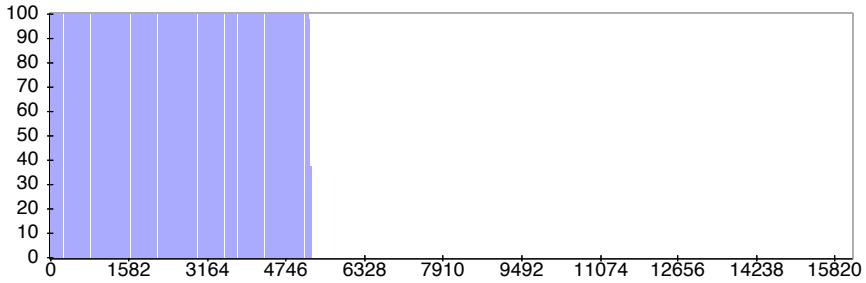
| | Counter | Map | Reduce | Total |
|---|---|---|---|---|
| Job Counters | SLOTS_MILLIS_MAPS | 0 | 0 | 454,162,761 |
| | Launched reduce tasks | 0 | 0 | 58 |
| | Total time spent by all reduces waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Rack-local map tasks | 0 | 0 | 7,938 |
| | Total time spent by all maps waiting after reserving slots | 0 | 0 | 0 |

# Some other time (March 2012)

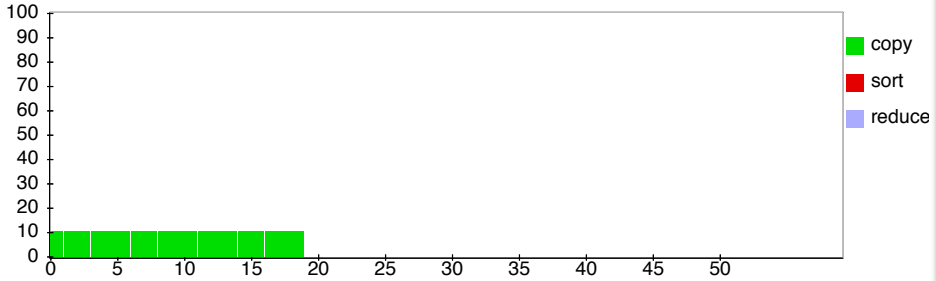
- Let's see what happened…

1h 16min

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 33.17% | 15816 | 10549 | 38 | 5229 | 0 | 0 / 0 |
| reduce | 4.17% | 50 | 31 | 19 | 0 | 0 | 0 / 0 |



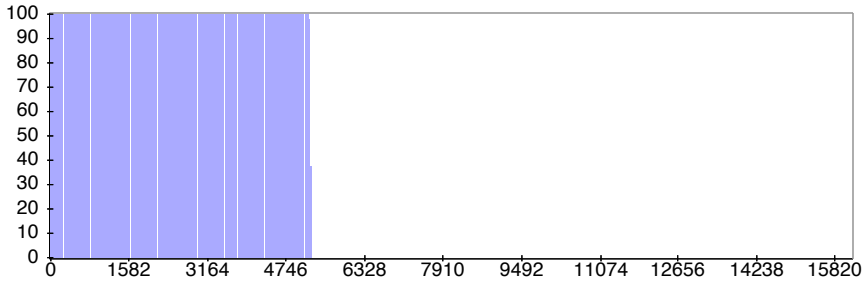

luce Completion Graph - close



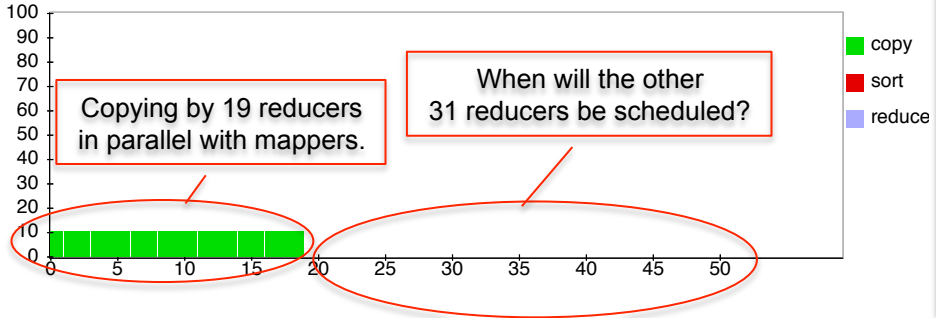
1h 16min

Only 19 reducers active, out of 50. Why?

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 33.17% | 15816 | 10549 | 38 | 5229 | 0 | 0 / 0 |
| reduce | 4.17% | 50 | 31 | 19 | 0 | 0 | 0 / 0 |



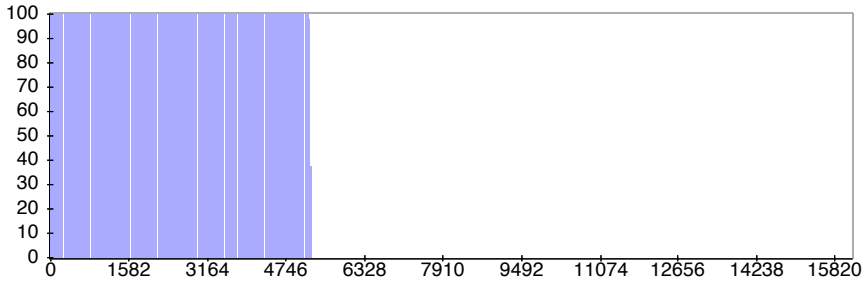

luce Completion Graph - close



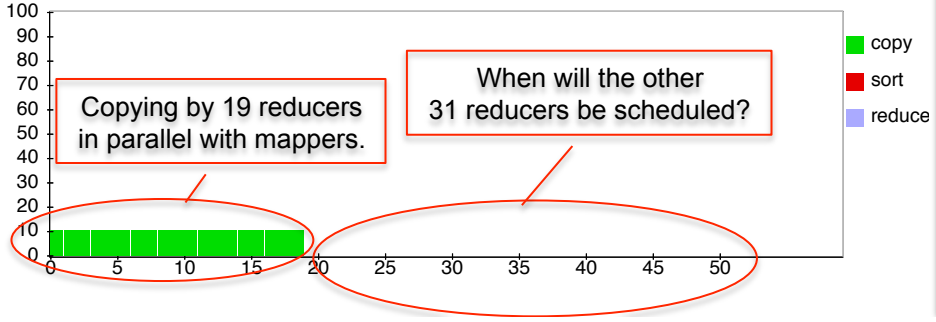
## 1h 16min

Only 19 reducers active, out of 50. Why?

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 33.17% | 15816 | 10549 | 38 | 5229 | 0 | 0 / 0 |
| reduce | 4.17% | 50 | 31 | 19 | 0 | 0 | 0 / 0 |

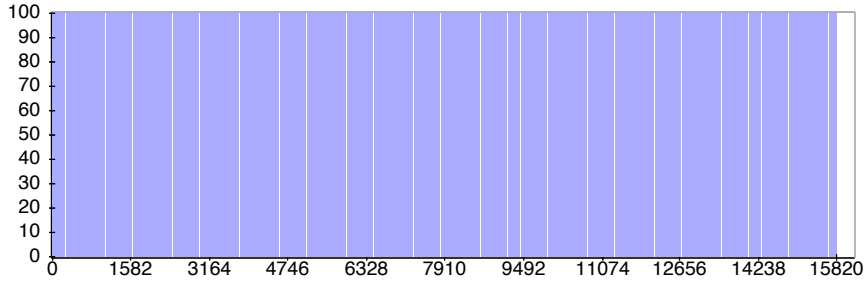



luce Completion Graph - close

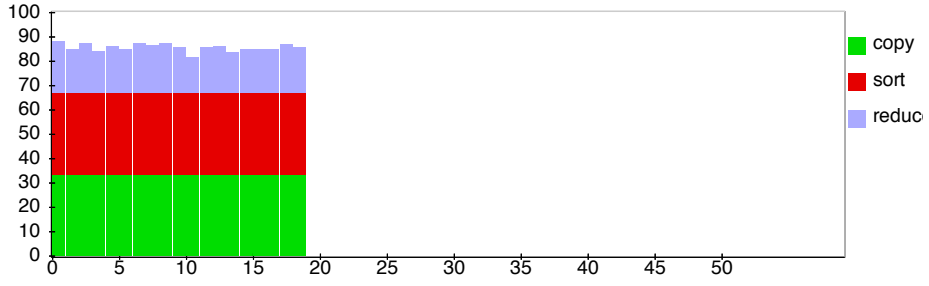



## 3h 50min

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 15816 | 0 | 0 | 15816 | 0 | 0 / 18 |
| reduce | 32.42% | 50 | 31 | 19 | 0 | 0 | 0 / 0 |



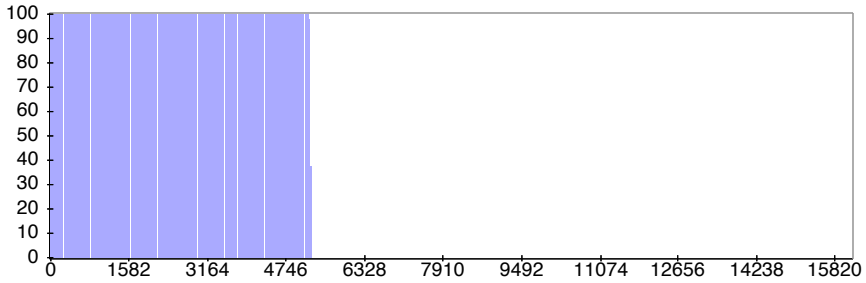

ıce Completion Graph - close



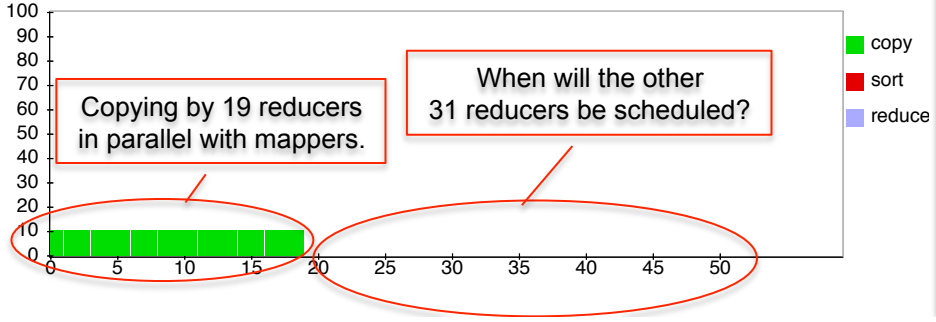
1h 16min

Only 19 reducers active, out of 50. Why?

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 33.17% | 15816 | 10549 | 38 | 5229 | 0 | 0 / 0 |
| reduce | 4.17% | 50 | 31 | 19 | 0 | 0 | 0 / 0 |




luce Completion Graph - close

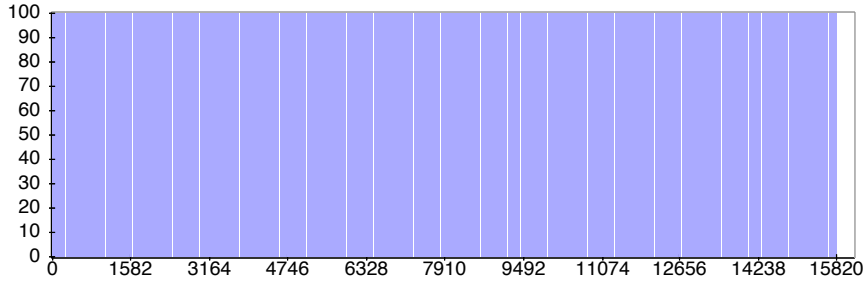



3h 50min

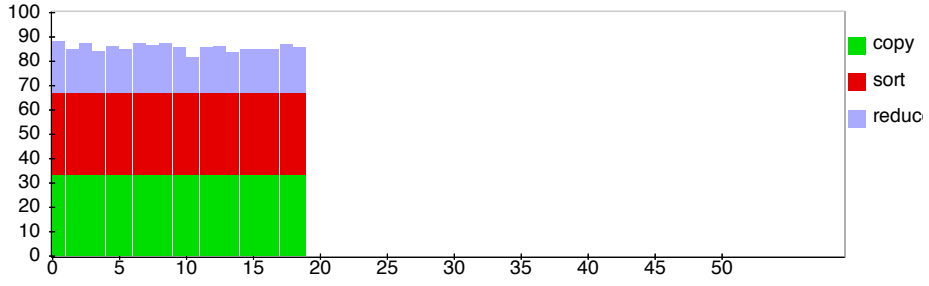
Speculative Execution

Completed. Sorting, and the rest of Reduce may proceed now

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 15816 | 0 | 0 | 15816 | 0 | 0 / 18 |
| reduce | 32.42% | 50 | 31 | 19 | 0 | 0 | 0 / 0 |



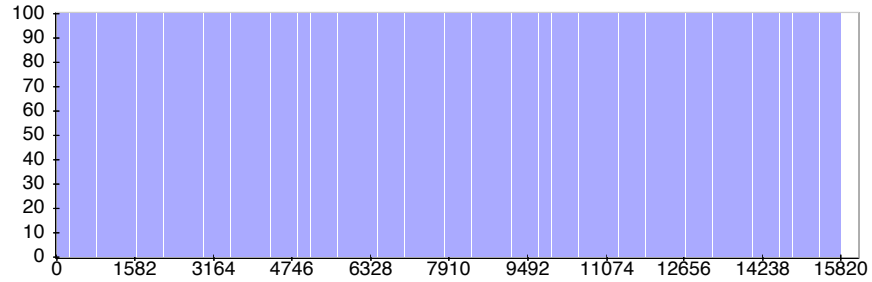

ıce Completion Graph - close



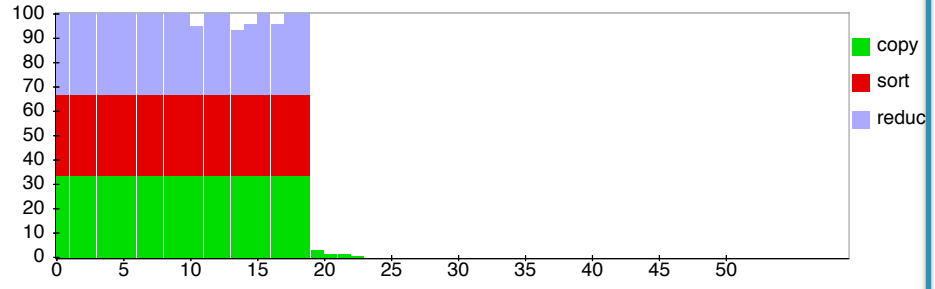
3h 51min

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 15816 | 0 | 0 | 15816 | 0 | 0 / 18 |
| reduce | 37.72% | 50 | 19 | 22 | 9 | 0 | 0 / 0 |

› Completion Graph - close



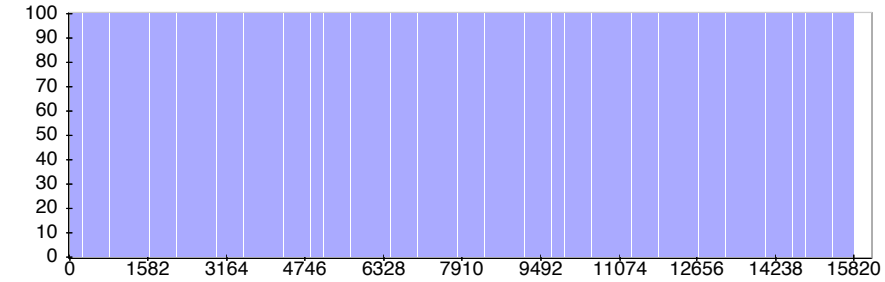

›uce Completion Graph - close



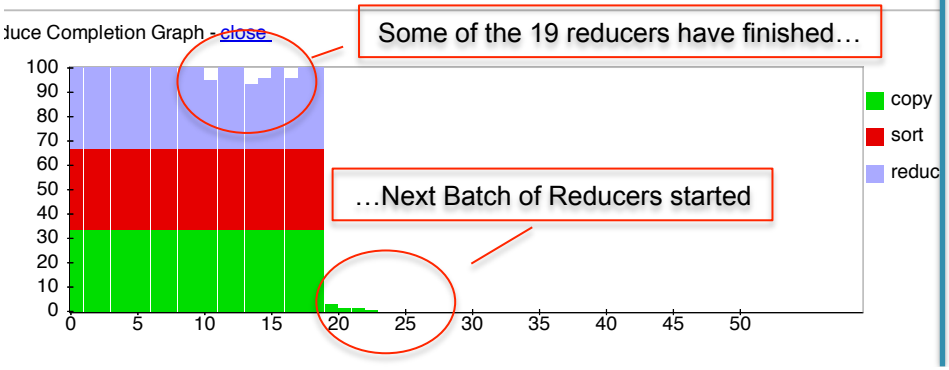
3h 51min

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 15816 | 0 | 0 | 15816 | 0 | 0 / 18 |
| reduce | 37.72% | 50 | 19 | 22 | 9 | 0 | 0 / 0 |

) Completion Graph - close



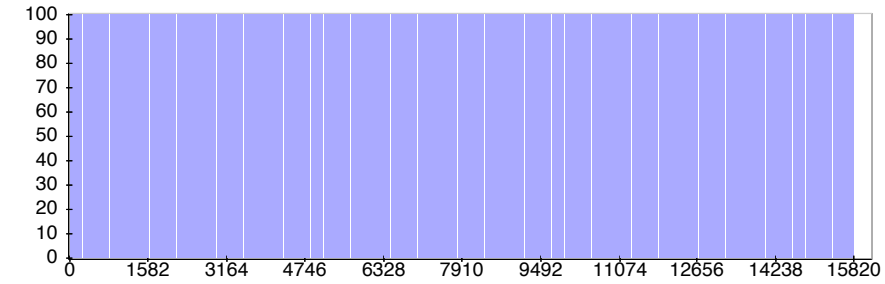

duce Completion Graph - close



## 3h 51min

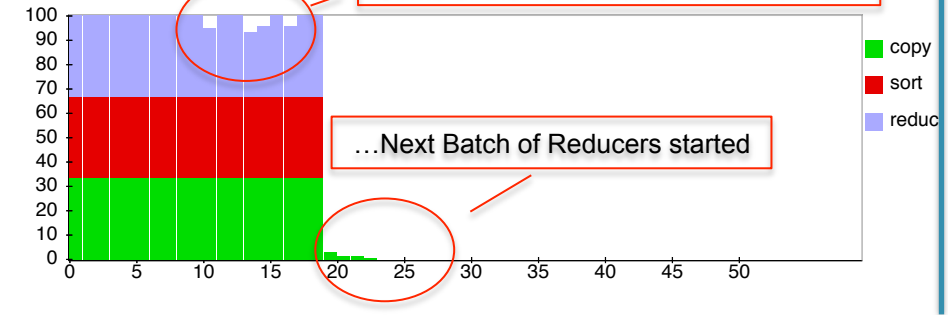
| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 15816 | 0 | 0 | 15816 | 0 | 0 / 18 |
| reduce | 37.72% | 50 | 19 | 22 | 9 | 0 | 0 / 0 |

) Completion Graph - close




duce Completion Graph - close

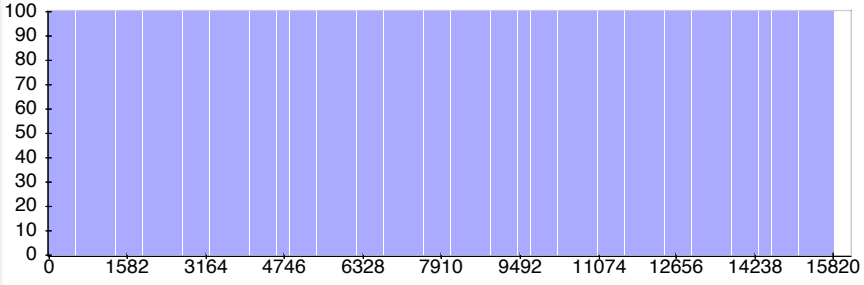
Some of the 19 reducers have finished…



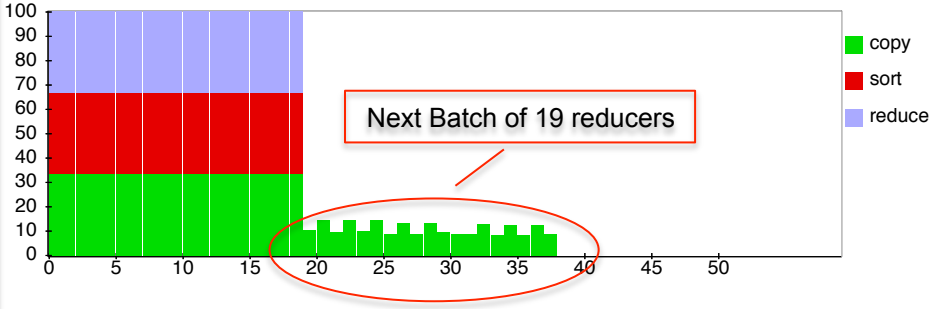

…Next Batch of Reducers started

## 3h 52min

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 15816 | 0 | 0 | 15816 | 0 | 0 / 18 |
| reduce | 42.35% | 50 | 11 | 20 | 19 | 0 | 0 / 0 |




ce Completion Graph - close

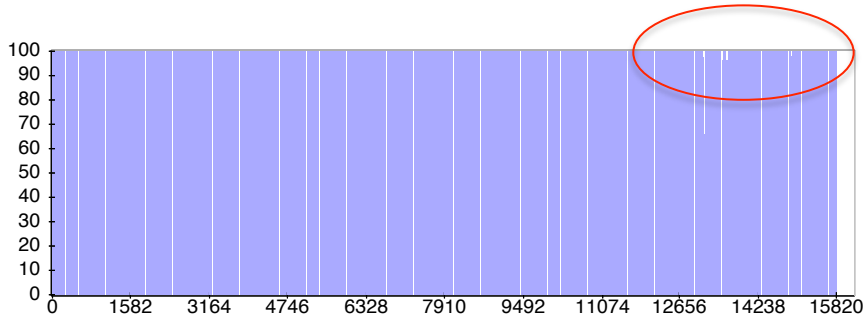



Next Batch of 19 reducers

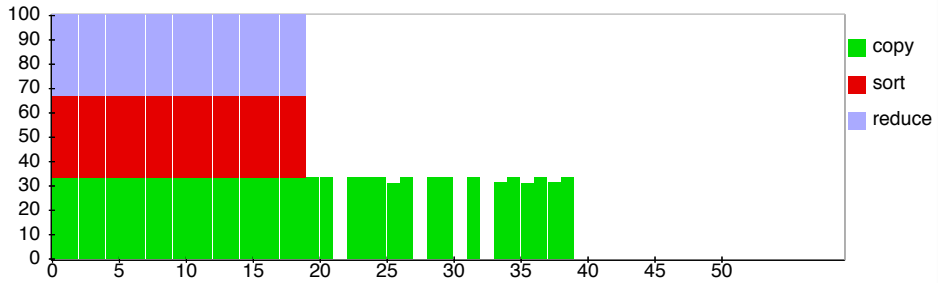
4h 18min

Several servers failed: “fetch error”. Their map tasks need to be rerun. All reducers are waiting….

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 99.88% | 15816 | 2638 | 30 | 13148 | 0 | 15 / 3337 |
| reduce | 48.42% | 50 | 15 | 16 | 19 | 0 | 0 / 0 |



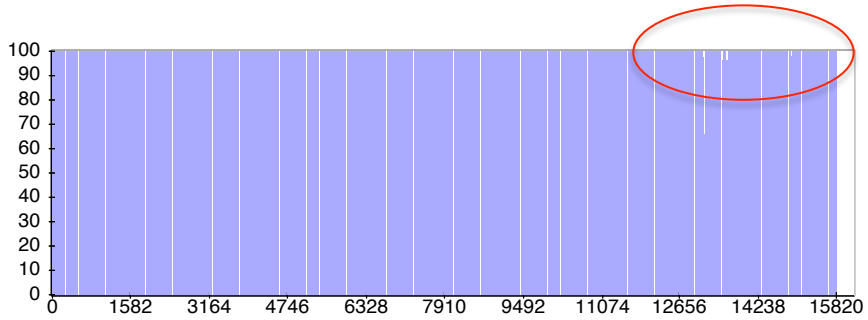

uce Completion Graph - close



4h 18min

Several servers failed: “fetch error”. Their map tasks need to be rerun. All reducers are waiting….

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 99.88% | 15816 | 2638 | 30 | 13148 | 0 | 15 / 3337 |
| reduce | 48.42% | 50 | 15 | 16 | 19 | 0 | 0 / 0 |

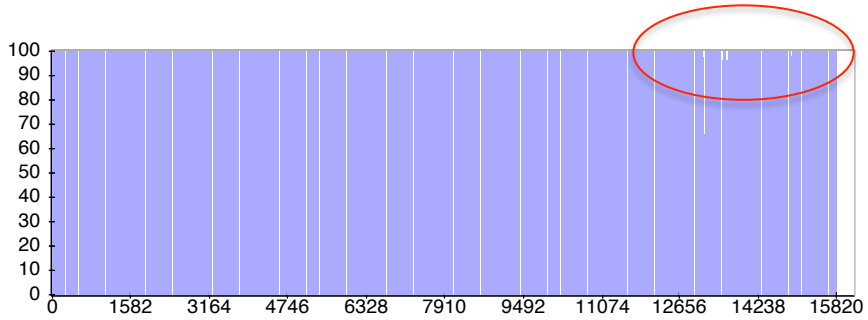



uce Completion Graph - close

## 4h 18min

Several servers failed: “fetch error”. Their map tasks need to be rerun. All reducers are waiting….

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 99.88% | 15816 | 2638 | 30 | 13148 | 0 | 15 / 3337 |
| reduce | 48.42% | 50 | 15 | 16 | 19 | 0 | 0 / 0 |

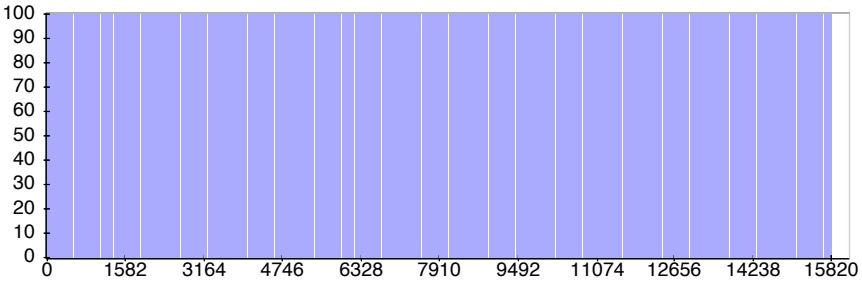



uce Completion Graph - close

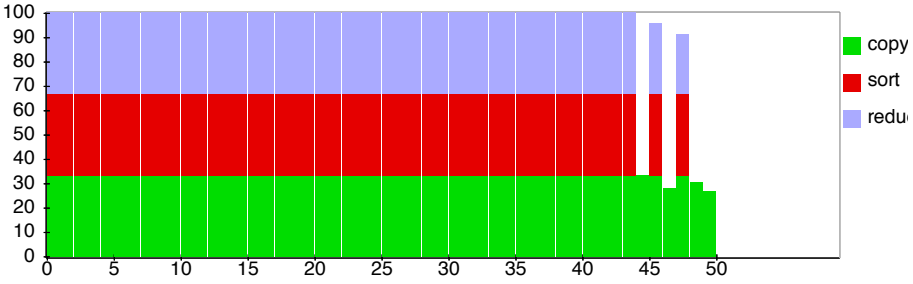

## 7h 10min

Mappers finished, reducers resumed.

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 15816 | 0 | 0 | 15816 | 0 | 26 / 5968 |
| reduce | 94.15% | 50 | 0 | 6 | 44 | 0 | 0 / 8 |




uce Completion Graph - close



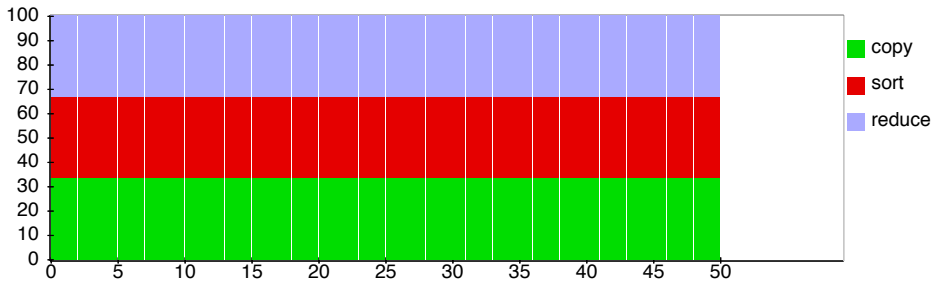
7h 20min

Success! 7hrs, 20mins.

# Hadoop job_201203041905_0001 on ip-10-203-30-146

**User:** hadoop
**Job Name:** PigLatin:DefaultJobName
**Job File:**
hdfs://10.203.30.146:9000/mnt/var/lib/hadoop/tmp/mapred/staging/hadoop/.staging/job_201203041905_0001/job.xml
**Submit Host:** ip-10-203-30-146.ec2.internal
**Submit Host Address:** 10.203.30.146
**Job-ACLs: All users are allowed**
**Job Setup:** Successful
**Status:** Succeeded
**Started at:** Sun Mar 04 19:08:29 UTC 2012
**Finished at:** Mon Mar 05 02:28:39 UTC 2012
**Finished in:** 7hrs, 20mins, 10sec
**Job Cleanup:** Successful
**Black-listed TaskTrackers:** 3

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 15816 | 0 | 0 | 15816 | 0 | 26 / 5968 |
| reduce | 100.00% | 50 | 0 | 0 | 50 | 0 | 0 / 14 |



# Pig Latin Mini-Tutorial

(will not discuss in detail in class; please read in order to do homework 8)

# Pig Latin Overview

- Data model = loosely typed *nested relations*
- Query model = a SQL-like, dataflow language

- Execution model:
  - Option 1: run locally on your machine; e.g. to debug
  - Option 2: compile into graph of MapReduce jobs, run on a cluster supporting Hadoop

# Example

- 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

Page(url, category, pagerank)

# First in SQL…

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

Page(url, category, pagerank)

# …then in Pig-Latin

```
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)
```

# Types in Pig-Latin

- Atomic: string or number, e.g. ‘Alice’ or 55
- Tuple: (‘Alice’, 55, ‘salesperson’)
- Bag: {(‘Alice’, 55, ‘salesperson’),
  (‘Betty’,44, ‘manager’), …}
- Maps: we will try not to use these

# Types in Pig-Latin

Tuple components can be referenced by number

- $0, $1, $2, …

Bags can be nested ! Non 1st Normal Form

- {(‘a’, {1,4,3}), (‘c’,{ }), (‘d’, {2,2,5,3,2})}

[Olston'2008]

$$t = \left('alice', \left\{ \begin{array}{c} ('lakers', 1) \\ ('iPod', 2) \end{array} \right\}, ['age' \rightarrow 20] \right)$$

Let fields of tuple t be called f1, f2, f3

| Expression Type | Example | Value for t |
|---|---|---|
| Constant | 'bob' | Independent of t |
| Field by position | $0 | 'alice' |
| Field by name | f3 | 'age' → 20 |
| Projection | f2.$0 | {('lakers'), ('iPod')} |
| Map Lookup | f3#'age' | 20 |
| Function Evaluation | SUM(f2.$1) | 1 + 2 = 3 |
| Conditional Expression | f3#'age'>18? 'adult':'minor' | 'adult' |
| Flattening | FLATTEN(f2) | 'lakers', 1<br>'iPod', 2 |

[Olston'2008]

# Loading data

- Input data = FILES !
  - Heard that before ?

- The LOAD command parses an input file into a bag of records
- Both parser (="deserializer") and output type are provided by user

For HW8: simply use the code provided

[Olston'2008]

# Loading data

```
queries = LOAD 'query_log.txt'
          USING userLoadFcn( )
          AS (userID, queryString, timeStamp)
```

Pig provides a set of built-in load/store functions

A = LOAD 'student' USING PigStorage('\t') AS (name: chararray, age:int, gpa: float);

same as

A = LOAD 'student' AS (name: chararray, age:int, gpa: float);

[Olston'2008]

# Loading data

- USING userfuction( ) -- is optional
  - Default deserializer expects tab-delimited file
- AS type – is optional
  - Default is a record with unnamed fields; refer to them as $0, $1, …
- The return value of LOAD is just a handle to a bag
  - The actual reading is done in pull mode, or parallelized

[Olston'2008]

# FOREACH

```
expanded_queries =
   FOREACH queries
   GENERATE userId, expandQuery(queryString)
```

expandQuery( ) is  a UDF that produces likely expansions
Note: it returns a bag, hence expanded_queries is a  nested bag

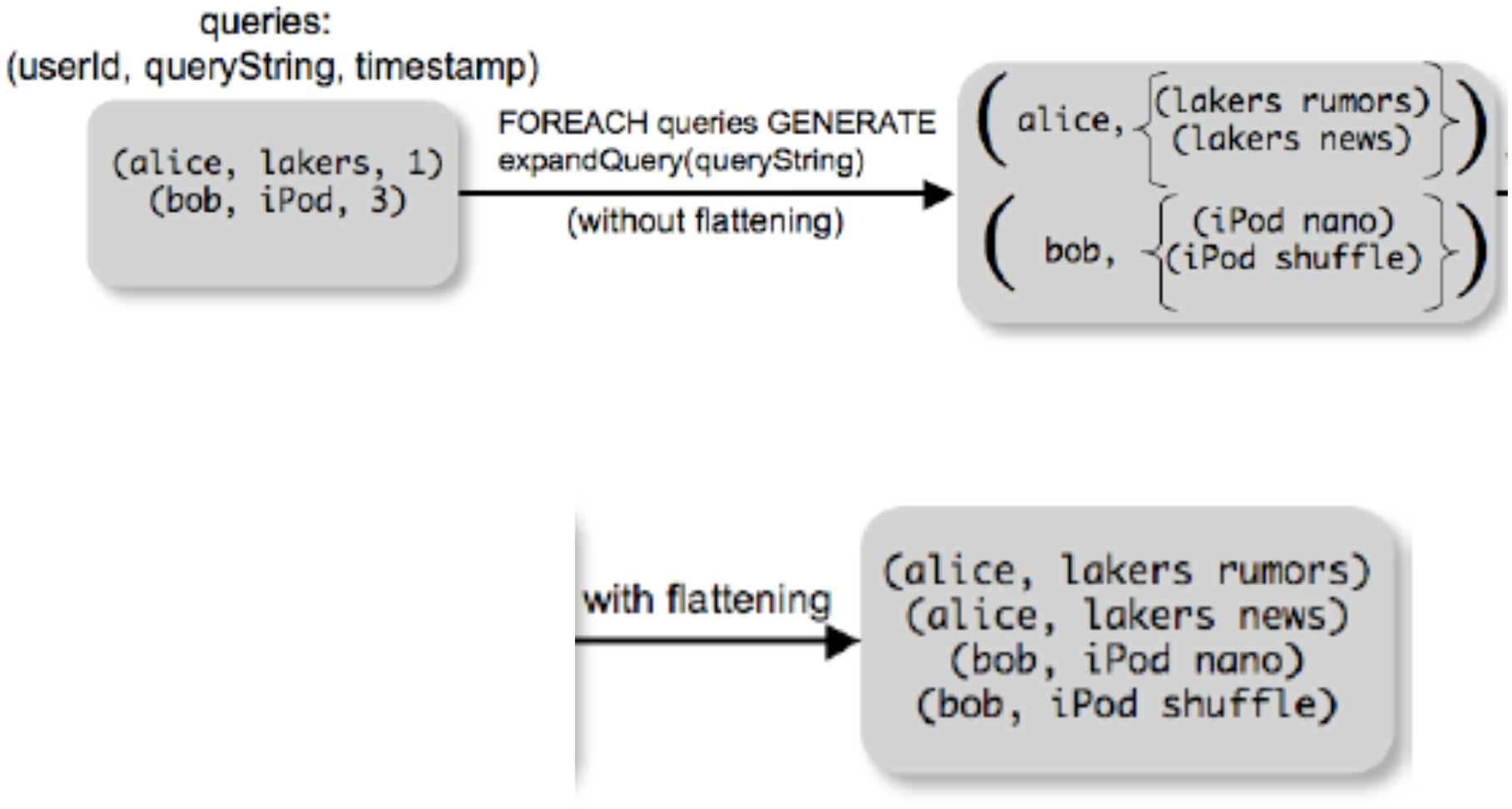
[Olston'2008]

# FOREACH

```
expanded_queries =
   FOREACH queries
   GENERATE userId,
                    flatten(expandQuery(queryString))
```

Now we get a flat collection

[Olston'2008]


queries:
(userId, queryString, timestamp)
(alice, lakers, 1)
(bob, iPod, 3)
FOREACH queries GENERATE
expandQuery(queryString)
(without flattening)
alice, {(lakers rumors) (lakers news)}
bob, {(iPod nano) (iPod shuffle)}
with flattening
(alice, lakers rumors)
(alice, lakers news)
(bob, iPod nano)
(bob, iPod shuffle)

[Olston'2008]

# FLATTEN

Note that it is NOT a normal function !

(that's one thing I don't like about Pig-latin)

- A normal FLATTEN would do this:
  - FLATTEN({{2,3},{5},{},{4,5,6}}) = {2,3,5,4,5,6}
  - Its type is: {{T}} → {T}
- The Pig Latin FLATTEN does this:
  - FLATTEN({4,5,6}) = 4, 5, 6
  - What is its Type? {T} → T, T, T, …, T ?????

[Olston'2008]

# FILTER

Remove all queries from Web bots:

```
real_queries =  FILTER queries BY userId neq 'bot'
```

Better: use a complex UDF to detect Web bots:

```
real_queries =  FILTER queries
                BY NOT isBot(userId)
```

[Olston'2008]

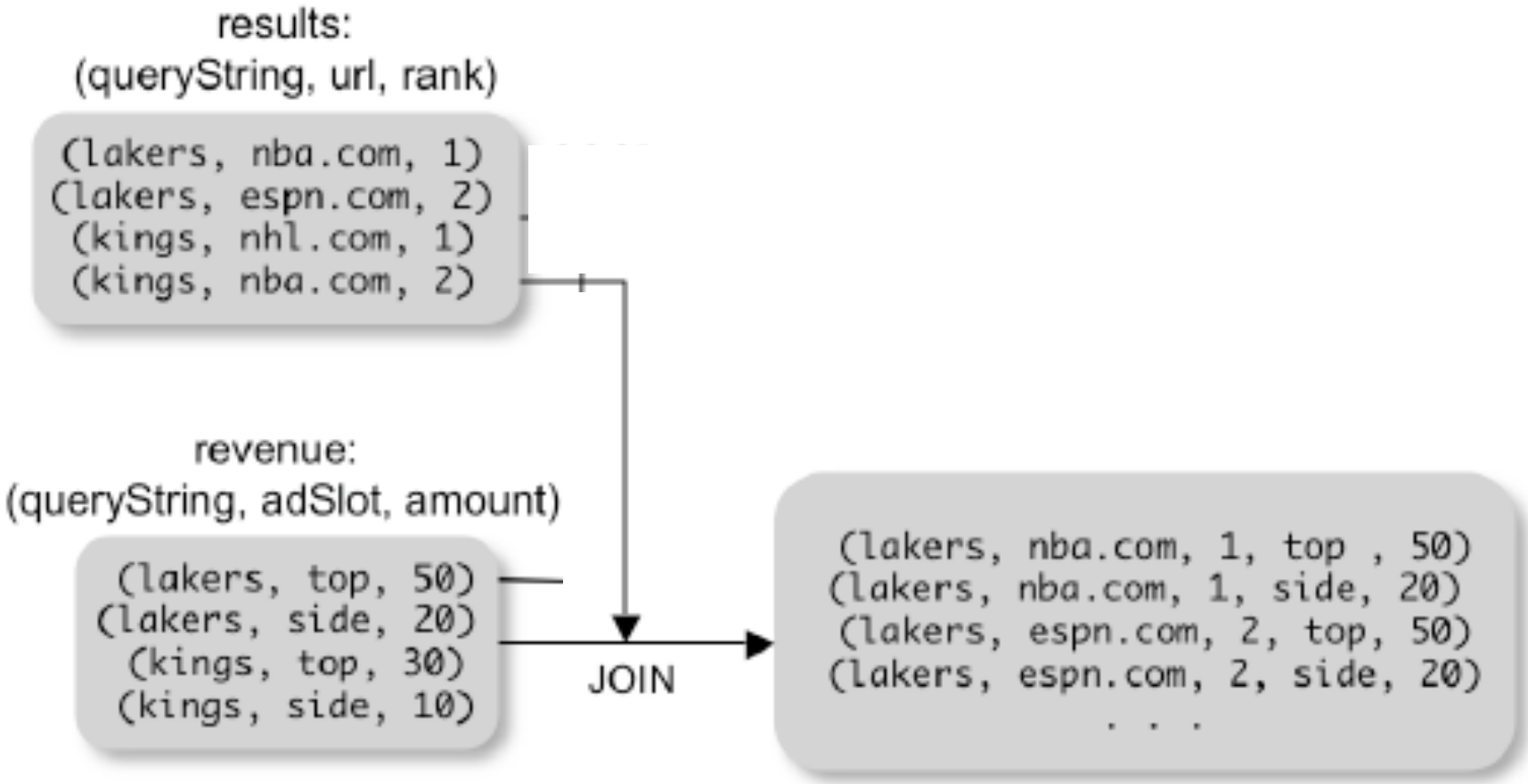
# JOIN

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

```
join_result = JOIN results BY queryString
                  revenue BY queryString
```

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

[Olston'2008]


results:
(queryString, url, rank)
(lakers, nba.com, 1)
(lakers, espn.com, 2)
(kings, nhl.com, 1)
(kings, nba.com, 2)
revenue:
(queryString, adSlot, amount)
(lakers, top, 50)
(lakers, side, 20)
(kings, top, 30)
(kings, side, 10)
JOIN
(lakers, nba.com, 1, top , 50)
(lakers, nba.com, 1, side, 20)
(lakers, espn.com, 2, top, 50)
(lakers, espn.com, 2, side, 20)
. . .

[Olston'2008]

# GROUP BY

revenue: {(queryString, adSlot, amount)}

```
grouped_revenue = GROUP revenue BY queryString
query_revenues =
    FOREACH grouped_revenue
    GENERATE queryString,
             SUM(revenue.amount) AS totalRevenue
```

grouped_revenue: {(queryString, {(adSlot, amount)})}
query_revenues: {(queryString, totalRevenue)}

[Olston'2008]

# Simple MapReduce

input : {(field1, field2, field3, . . . .)}

```
map_result = FOREACH input
                 GENERATE FLATTEN(map(*))
key_groups = GROUP map_result BY $0
output = FOREACH key_groups
         GENERATE $0, reduce($1)
```

map_result : {(a1, a2, a3, . . .)}
key_groups : {(a1, {(a2, a3, . . .)})}

[Olston'2008]

# Co-Group

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

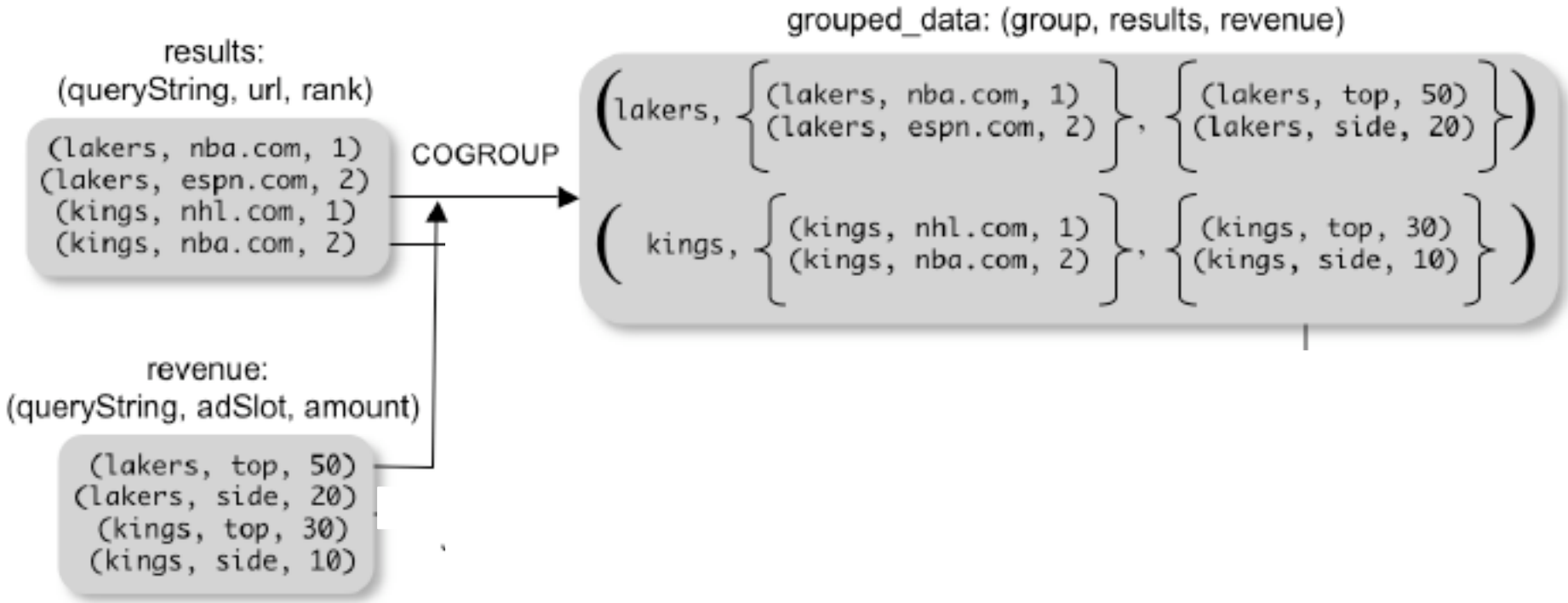
```
grouped_data =
      COGROUP results BY queryString,
              revenue BY queryString;
```

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

What is the output type in general ?

[Olston'2008]

# Co-Group




Is this an inner join, or an outer join ?

[Olston'2008]

# 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.

[Olston'2008]

# Co-Group v.s. Join

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

```
grouped_data = COGROUP results BY queryString,
                       revenue BY queryString;
join_result = FOREACH grouped_data
              GENERATE FLATTEN(results),
                       FLATTEN(revenue);
```

Result is the same as JOIN

[Olston'2008]

# Asking for Output: STORE

```
STORE query_revenues INTO `theoutput'
        USING userStoreFcn();
```

Meaning: write query_revenues to the file 'theoutput'

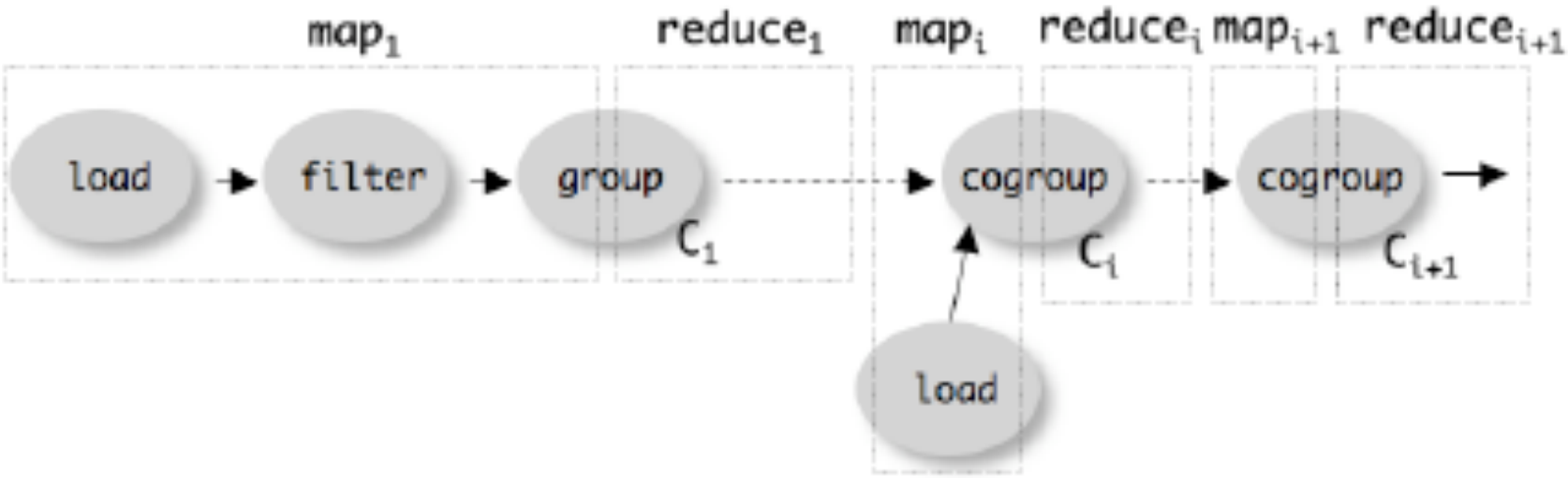
[Olston'2008]

# Implementation

- Over Hadoop !
- Parse query:
  - Everything between LOAD and STORE → one logical plan
- Logical plan → graph of MapReduce ops
- All statements between two (CO)GROUPs → one MapReduce job

[Olston'2008]

# Implementation



# Review: MapReduce

- Data is typically a file in the Google File System
  - HDFS for Hadoop
  - File system partitions file into chunks
  - Each chunk is replicated on k (typically 3) machines
- Each machine can run a few map and reduce tasks simultaneously
- Each map task consumes one chunk
  - Can adjust how much data goes into each map task using “splits”
  - Scheduler tries to schedule map task where its input data is located
- Map output is partitioned across reducers
- Map output is also written locally to disk
- Number of reduce tasks is configurable
- System shuffles data between map and reduce tasks
- Reducers sort-merge data before consuming it

# MapReduce Phases