SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Established in collaboration with MIT

Big data

PROF. D. HERREMANS

PART2 - LAB

50.038 Computational Data Science

Hadoop & MapReduce

1. **(easy way) through a virtual machine image**

2. Native install on your Unix machine

   - Installation instructions, see https://goo.gl/c6HDTt

- We will use Hadoop Streaming, which will allow us to use Python for MapReduce scripts

Hadoop filesystem (fs) commands

List files:

```
hadoop fs -ls
```

Store a local file in hdfs:

```
hadoop fs -put myfile.txt
```

Rename file:

```
hadoop fs -mv myfile.txt newname.txt
```

Remove file:

```
hadoop fs -rm filename.txt
```

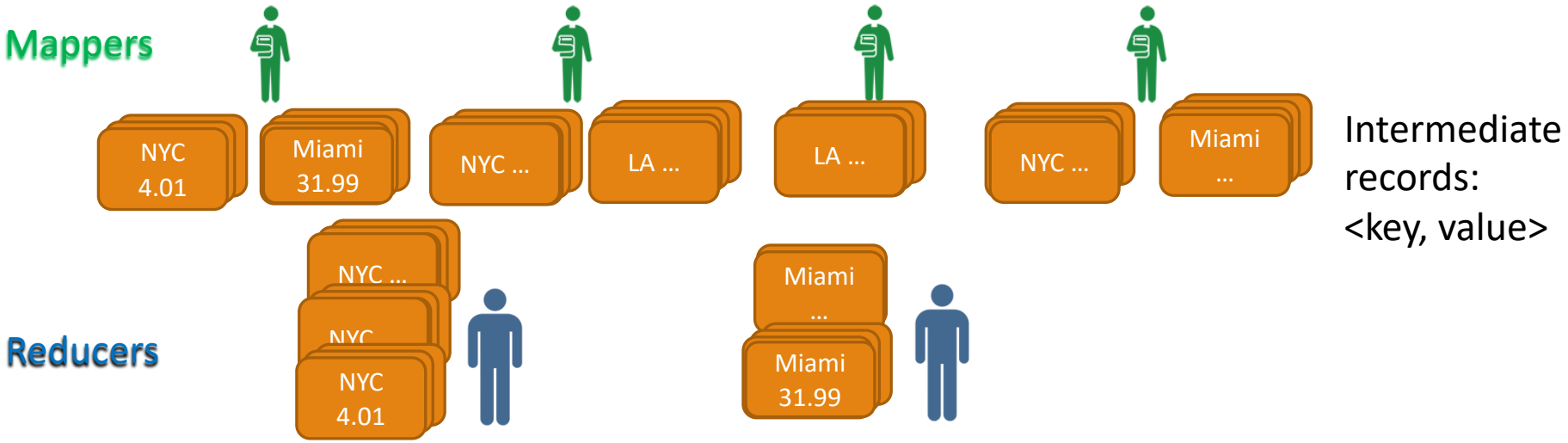
Create directory:

```
hadoop fs -mkdir newdir
```

=> very similar to traditional unix commands

MapReduce

- What if you had more people to help? Mappers and Reducers
- Break ledger into chucks and distribute to mappers



- Reducers get assigned a store
- They ask for the stack of that store at each Mapper: shuffle
- They alphabetically go through their stacks: sort; and process them

Example

- Searching for: total number of sales per store
- Input data (purchases.txt in the folder data):

```
2012-12-31      17:59   Birmingham      CDs     118.04  Cash
2012-12-31      17:59   Las Vegas       Health and Beauty       420.46  Amex
2012-12-31      17:59   Wichita Toys    383.9   Cash
2012-12-31      17:59   Tucson  Pet Supplies    268.39  MasterCard
2012-12-31      17:59   Glendale        Women's Clothing        68.05   Amex
2012-12-31      17:59   Albuquerque     Toys    345.7   MasterCard
2012-12-31      17:59   Rochester       DVDs    399.57  Amex
2012-12-31      17:59   Greensboro      Baby    277.27  Discover
2012-12-31      17:59   Arlington       Women's Clothing        134.95  MasterCa
rd
```

Note you will need to 'put' the file on the Hadoop filesystem first (it's in the data folder):
Navigate to the correct folder:
cd udacity_training [use tab to autocomplete]
hadoop fs –put data/purchases.txt [because it is in the data/ folder]

Uploading data files to hdfs

Note you will need to 'put' the file on the Hadoop filesystem first. It is in your local disc's data folder. Navigate to the correct folder:

- cd udacity_training [use tab to autocomplete]
- hadoop fs –put data/purchases.txt [because it is in the data/ folder]

Create a folder for your input data on the hdfs:

- hadoop fs—mkdir myinput

Move the data file to this folder:

- hadoop fs –mv purchases.txt myinput

Example 1: total sales per store

Mapper.py

```
# date\ttime\tstore name\titem description\tcost\tmethod of payment
# We want elements 2 (store name) and 4 (cost)
# We need to write them out to standard output, separated by a tab

import sys
for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, store, item, cost, payment = data
        print "{0}\t{1}".format(store, cost)
```

Example 1: total sales per store

Reducer.py

Loop around the data - # It will be in the format key\tval, where key is the store name, val is the sale amount. All the sales for a particular store will be presented, then the key will change and we'll be dealing with the next store.

```
Import sys
oldKey = None
 salesTotal = 0
for line in sys.stdin:
    data_mapped = line.strip().split("\t")
    if len(data_mapped) != 2:
        # Something has gone wrong. Skip this line.
        continue
```

```
Miami    12.34
Miami    99.07
Miami    3.14
NYC      99.77
NYC      88.99
```

Example 1: total sales per store

Reducer.py …

```
    thisKey, thisSale = data_mapped
    if oldKey and oldKey != thisKey:
        print oldKey, "\t", salesTotal
        oldKey = thisKey;
        salesTotal = 0
    oldKey = thisKey
    salesTotal += float(thisSale)

if oldKey != None:
    print oldKey, "\t", salesTotal
```

Upload python code to hdfs

Assuming you are in the udacity_training folder (check using pwd command (print working directory))

Put your reducer.py and mapper.py function in udacity_training/code/

- hadoop fs –put code/mapper.py
- hadoop fs –put code/reducer.py

Running MapReduce tasks

```
hadoop jar /usr/lib/hadoop….hadoopstreaming.jar –mapper
mapper.py –reducer reducer.py –file mapper.py –file
reducer.py—input myinput –output joboutput
```

Or quicker:

```
hs mapper.py reducer.py myinput joboutput
```

Make sure you are in the code/ folder (cd ~udacity_training/code/ or add the full url to mapper.py and reducer.py)

Job tracker: http://localhost:50030/jobtracker.jsp

Note: myinput is the folder in which purchases.txt is stored on your hd. fs

Important: output directory cannot exist yet

Ex 1: defensive mapper code

```
def mapper():
  for line in sys.stdin:
    data = line.strip().split("\t")
     # This is the place you need to do some defensive programming
     # what if there are not exactly 6 fields in that line?
     # YOUR CODE HERE
     date, time, store, item, cost, payment = data
     print "{0}\t{1}".format(store, cost)
```

Ex 1: defensive mapper code

```
def mapper():
  for line in sys.stdin:
    data = line.strip().split("\t")
     # This is the place you need to do some defensive programming
     # what if there are not exactly 6 fields in that line?
     if len(data) == 6:
        date, time, store, item, cost, payment = data
        print "{0}\t{1}".format(store, cost)
```

Results of MapReduce tasks

```
hadoop fs -ls joboutput
```

```
Found 3 items
-rw-r--r--   1 training supergroup          0 2013-09-12 21:24 joboutput/_SUCCESS
drwxr-xr-x   - training supergroup          0 2013-09-12 21:23 joboutput/_logs
-rw-r--r--   1 training supergroup       2296 2013-09-12 21:24 joboutput/part-00000
```

```
hadoop fs -cat joboutput/part-00000 | less
```

```
Anaheim          10076416.36
Anchorage        9933500.4
Arlington        10072207.97
Atlanta          9997146.7
Aurora  9992970.92
Austin  10057158.9
Bakersfield      10031208.92
Baltimore        10096521.45
Baton Rouge      10131273.23
Birmingham       10076606.52
Boise   10039166.74
Boston  10039473.28
```

-> Total sales per store

Results of MapReduce tasks

Finding a particular city:

```
hadoop fs –cat joboutput/part-00000 | grep Spokane
```

>>Spokane 10083362.98

```
hadoop fs –get joboutput/part-00000 mylocalfile
```

→ copies it to your local disk (opposite of 'put')

MR Design Patterns

- Summarization patters:
  - Reverse index: to allow for faster searching (e.g. back of a book)
  - Numerical summarizations: sum, average, min/max, counting, statistics…
- Filtering patterns:
  - Simple filter
  - Top-N: each mapper generates top-n, reducers do the same
  - Sampling
- Structural patterns:
  - Combining data sets (e.g. when migrating from RDBMS to Hadoop-based)

For a quick lab check-off

- Keep screenshots of what you are doing in a Word document.
- Paste your code in there too.
- This will allow us to see your progress more quickly!

Lab 2 – exercise 1

- Run the previous example on a virtual machine on your own machine
  - In particular: start by getting the total sales for each city.
  - Before you begin: make sure to:
    - Create a directory ‘myinput’ on your hadoop filesystem.
    - Upload (put) purchases.txt on your hadoop filesystem.
    - Move purchases.txt to the myinput folder
- Then move on to the exercises in the next slides.

Exercises 2-4

2. Instead of breaking the sales down by store, instead retrieve a sales breakdown by **product category** across all of our stores (instead of by city).
3. Find the monetary value for the **highest individual sale for each separate store**.
4. Find the **total sales value across all the stores**, and the **total number of sales**. Assume there is only one reducer.

Quick check for answers

2. Total sales for Toys = 57,463,477.11
3. Highest individual sale for Reno = 499.99
4. Number of sales = 4,138,476, with total value = 1,034,457,953.26

Advanced exercise 5

Common log file: use **access_log** as input

10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET /assets/js/lowpro.js HTTP/1.1" 200 10469 %h %l %u %t \"%r\" %>s %b

%h is the IP address of the client

%l is identity of the client, or "-" if it's unavailable

%u is username of the client, or "-" if it's unavailable

%t is the time that the server finished processing the request. The format is [day/month/year:hour:minute:second zone]

%r is the request line from the client is given (in double quotes). It contains the method, path, query-string, and protocol or the request.

%>s is the status code that the server sends back to the client. You will see see mostly status codes 200 (OK - The request has succeeded), 304 (Not Modified) and 404 (Not Found). See more information on status codes in W3C.org

%b is the size of the object returned to the client, in bytes. It will be "-" in case of status code 304.

You'll have to write new Mappers and Reducers

Exercise 5-7

5. Write a MapReduce program to find number of hits for each different file on the Web site.
6. Write a MapReduce program which determines the number of hits to the site made by each different IP address.
7. Find the most popular file on the website, i.e. whose file path occurs most often in access_log. Your reducer should output the file's path and number of times it appears in the log

Quick check for answers

- Hit for the page /assets/js/the-associates.js = 2456
- Hits made by IP address 10.99.99.186 = 6
- Most popular file's pathname = combined.css
  Number of occurrences = 117348

- *(tip, use grep to see this)*