15-319 / 15-619 Cloud Computing

Recitation 12

November 17th and 19th , 2015

Overview

- **Administrative issues**
  - Tagging, 15619Project, project code
- **Last week's reflection**
  - Project 3.5
  - Quiz 10
- **This week's schedule**
  - Project 4.1, Batch Processing with MapReduce
  - Unit 5 - Module 19
  - 15619Project Phase 3
  - Quiz 11
- **Demo**
- **Twitter Analytics: The 15619Project**

Reminders

- Monitor AWS expenses regularly and tag all resources
  - Check your bill (Cost Explorer > filter by tags).
- Piazza Guidelines
  - Please tag your questions appropriately
  - Search for an existing answer first
- Provide clean, modular and well documented code
  - Large penalties for not doing so.
  - **Double check** that your code is submitted!! (verify by downloading it from TPZ from the submissions page)
- Utilize Office Hours
  - We are here to help (but not to give solutions)
- Use the team AWS account and tag the 15619Project resources carefully

Project 3.5 : FAQs

Problem 1: Out-of-memory issue during partitioning

- Should make sure the partition is really necessary
- Partitioning on a big table may drain the datanode's memory

Problem 2: Invalid File Footer

- Happens when specified compression encoding is different from the actual compression encoding
- Should find some way to convert uncompressed data into compressed format, e.g. INSERT INTO tb_opt SELECT * from tb

Module to Read

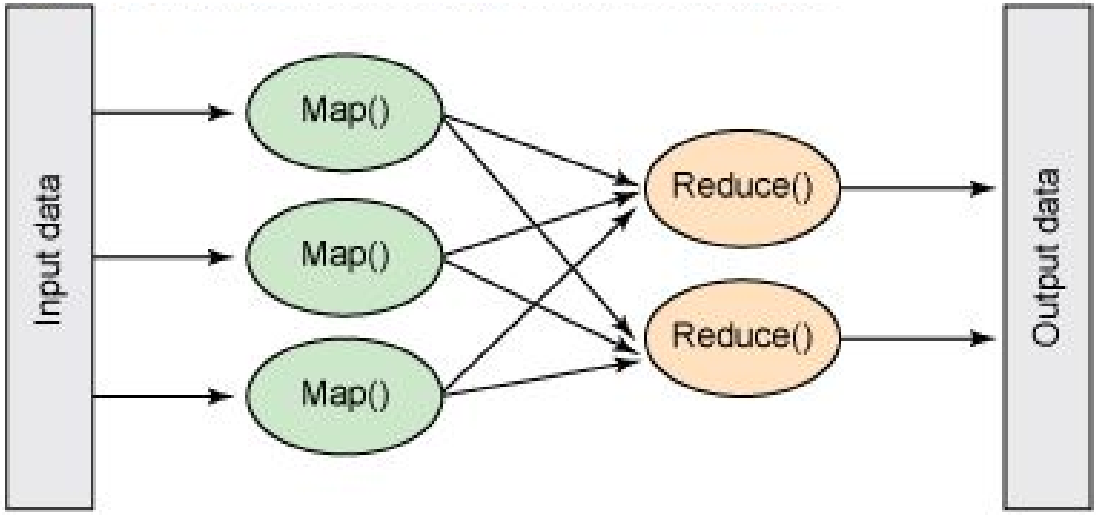
- UNIT 5: Distributed Programming and Analytics Engines for the Cloud
  - Module 18: Introduction to Distributed Programming for the Cloud
  - Module 19: Distributed Analytics Engines for the Cloud: MapReduce
    - Hadoop 1.0
    - Hadoop 2.0 - YARN
  - Module 20: Distributed Analytics Engines for the Cloud: Spark
  - Module 21: Distributed Analytics Engines for the Cloud: GraphLab

Project 4

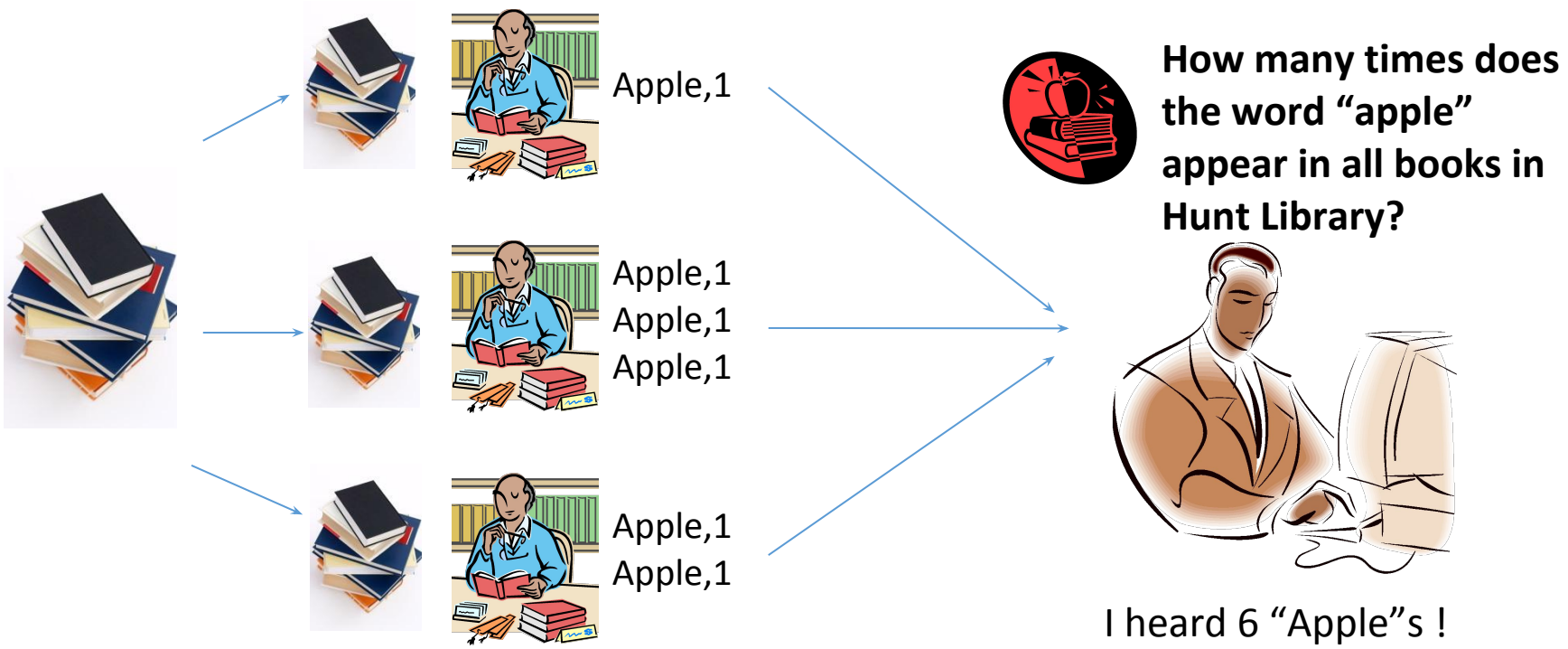
- Project 4.1, Batch Processing with MapReduce
  - MapReduce Programming Using YARN
- Project 4.2
  - Iterative Programming Using Apache Spark
- Project 4.3
  - Stream Processing using Kafka/Samza

Introduction to MapReduce

- **Definition**: Programming model for processing large data sets with a parallel, distributed algorithm on a cluster
- **Map**: Extract something you care about
- **Group by key**: Sort and Shuffle
- **Reduce**: Aggregate, summarize, filter or transform
- **Output** the result

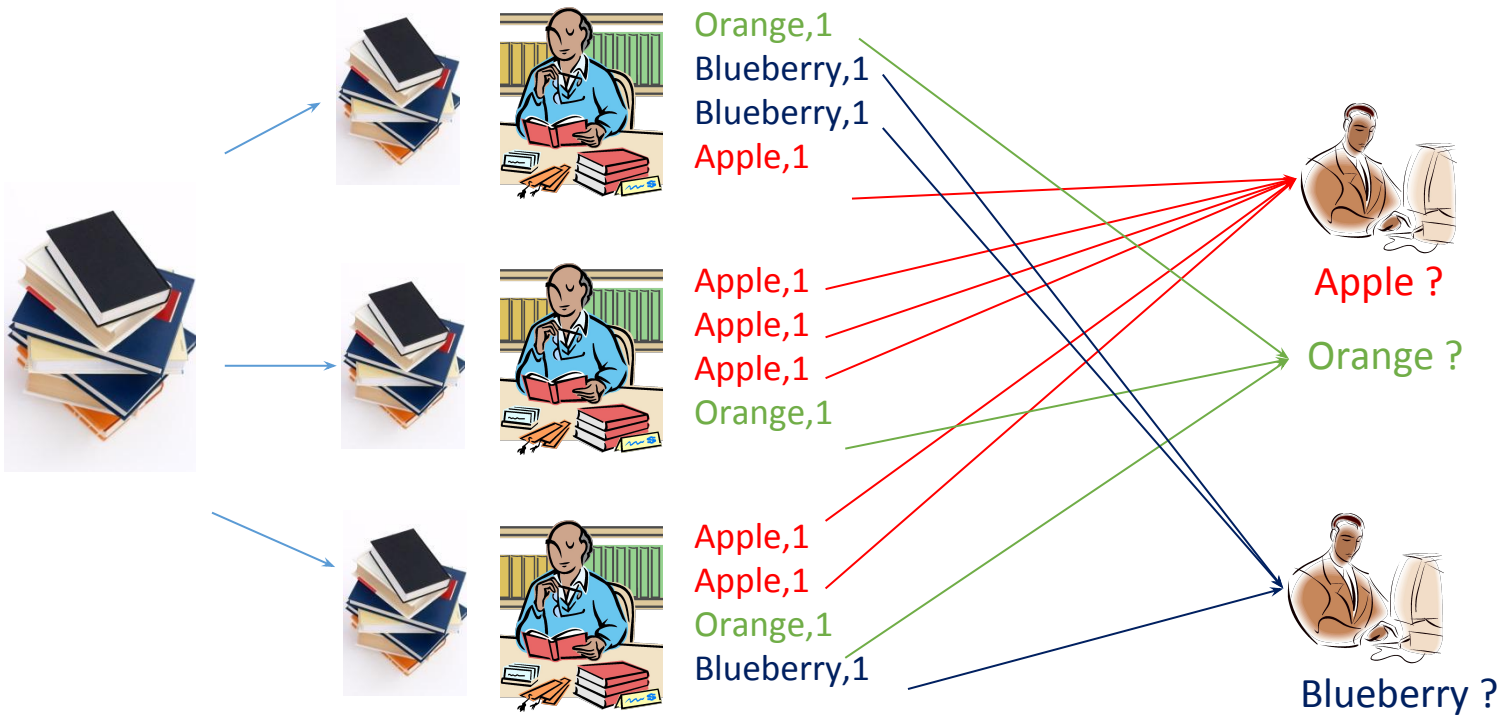


MapReduce - Introduced in Project 1



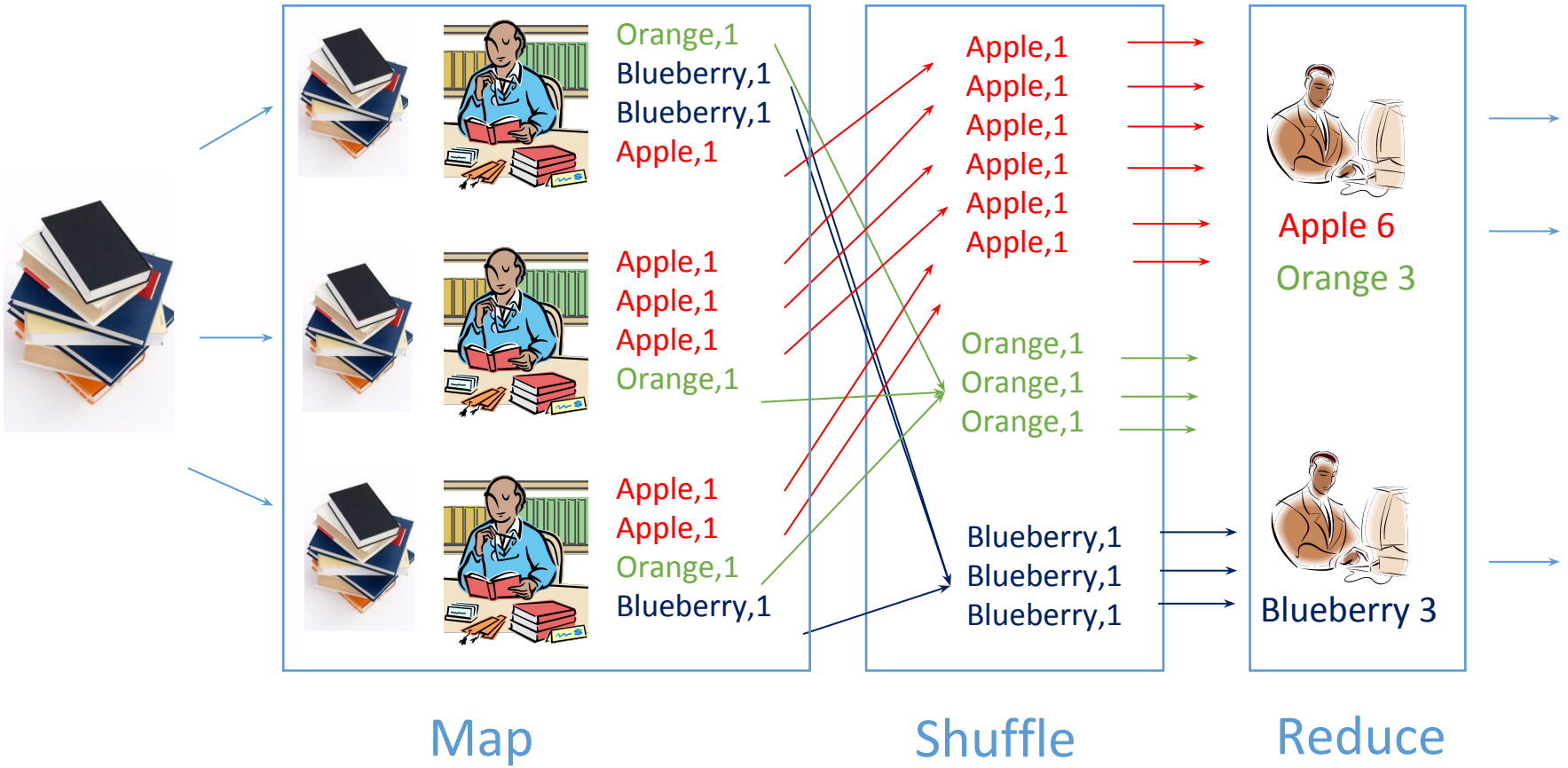
MapReduce Example

What if we want to count the number of times all fruits appeared in these books?

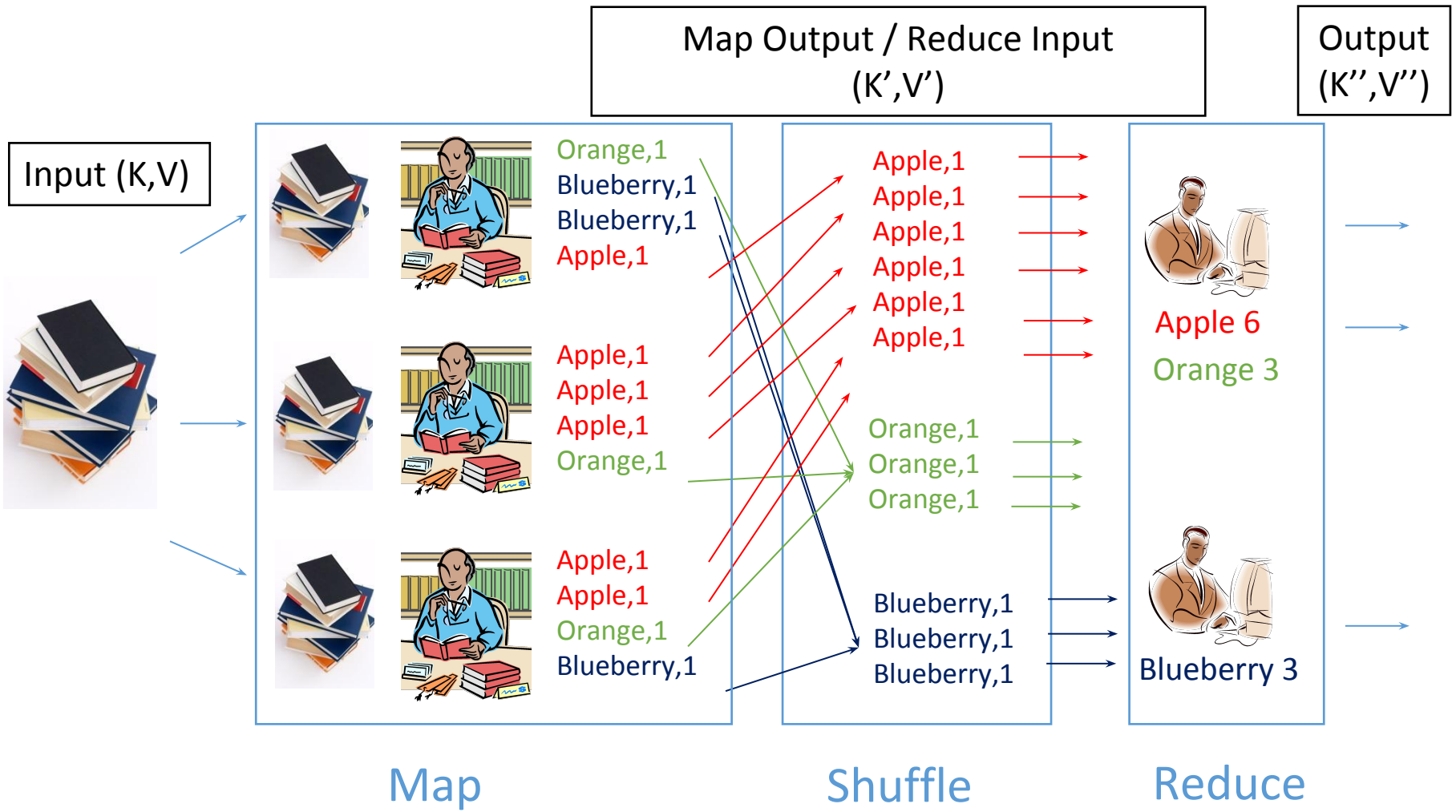


You can have multiple aggregators, each one working on a distinct set of “fruits”.

MapReduce Example



MapReduce Example



Steps of MapReduce

- Map
- Shuffle
- Reduce
- Produce final output

Steps of MapReduce

- Map
  - Prepare input for mappers
    - Split input into parts and assign them to mappers
  - Map Tasks
    - Each mapper will work on its portion of the data
    - Output: **key-value pairs**
      - Keys are used in Shuffling and Merge to find the Reducer that handles it
      - Values are messages sent from mapper to reducer
      - e.g. (Apple, 1)

Steps of MapReduce

- Shuffle
  - Sort and group by key:
    - Split keys and assign them to reducers (based on hashing)
    - Each key will be assigned to exactly one reducer
- Reduce
  - Input: mapper's output (key-value pairs)
  - Each reducer will work on one or more keys
  - Output: the result needed
- Produce final output
  - Collect all output from reducers
  - Sort them by key

MapReduce - Data Types

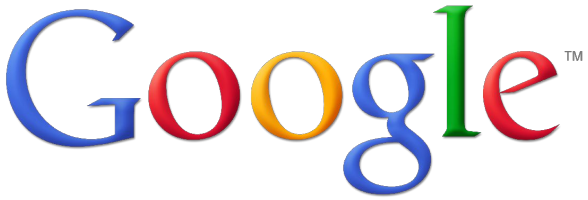
- Mapper (default)
  - Input: **key-value pairs**
    - **Key:** byte offset of the line
    - **Value:** the text content of the line
  - Output: **key-value pairs**
    - **Key:** specified by your program
    - **Value:** specified by your program based on what content you expect the reducer to receive as a list

(k1,v1) -> Mapper -> (k2,v2)

MapReduce - Data Types

- Reducer
  - Input: **key-value pairs**
    - A list of values for each key output from the mapper
  - Output: **key-value pairs**
    - The desired result from your aggregation

(k2,list(v2)) -> Reducer -> (k3,v3)



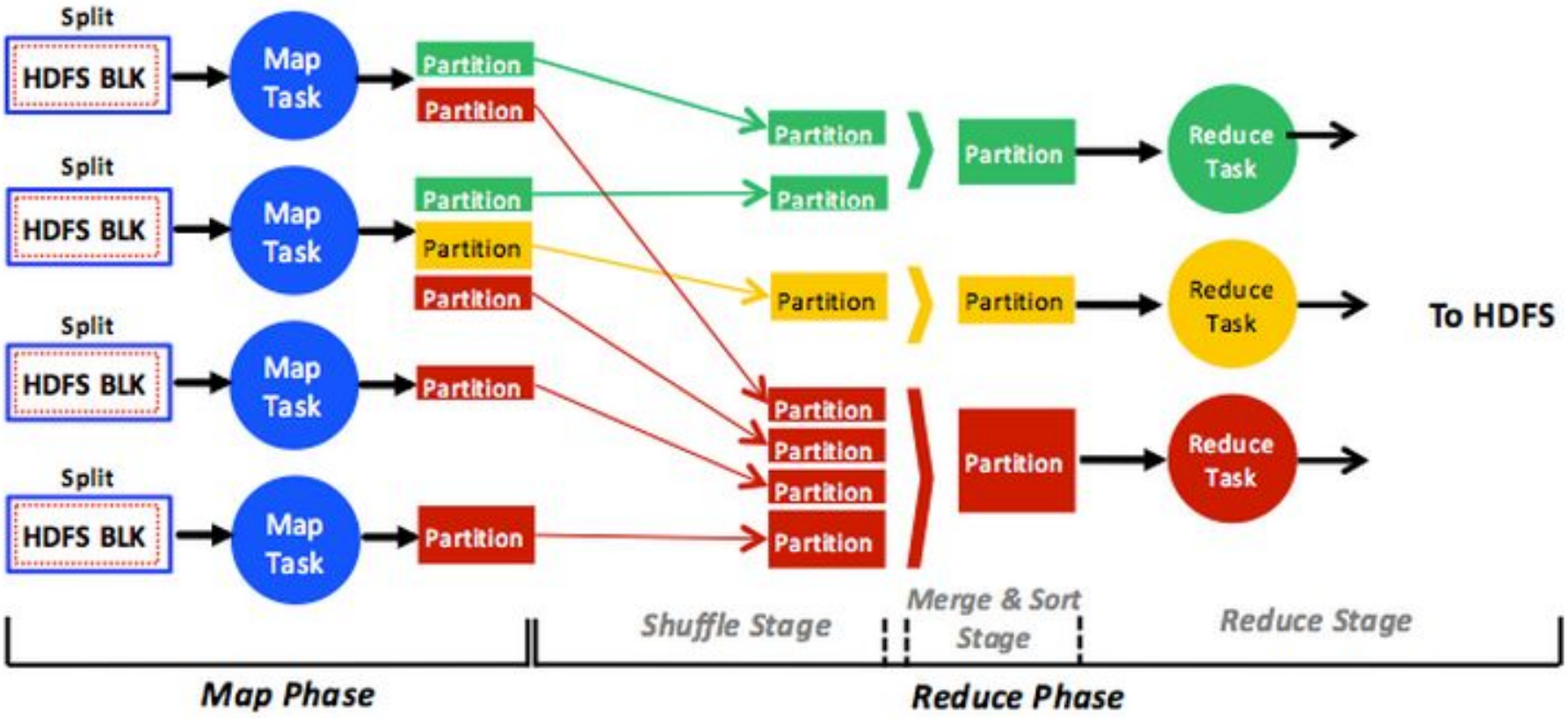
Proprietary		Open Source
GFS	→	HDFS
MapReduce	→	MapReduce
BigTable	→	HBase

Hadoop

- MapReduce
  - A programming model for processing large data sets using a parallel distributed algorithm
- Apache Hadoop
  - A framework for running MapReduce applications on a large cluster of commodity hardware
  - Implements the MapReduce computational paradigm
  - Uses HDFS for data storage
  - Engineers with little knowledge of distributed computing can write the code in a short period

MapReduce and HDFS

- Detailed workflow

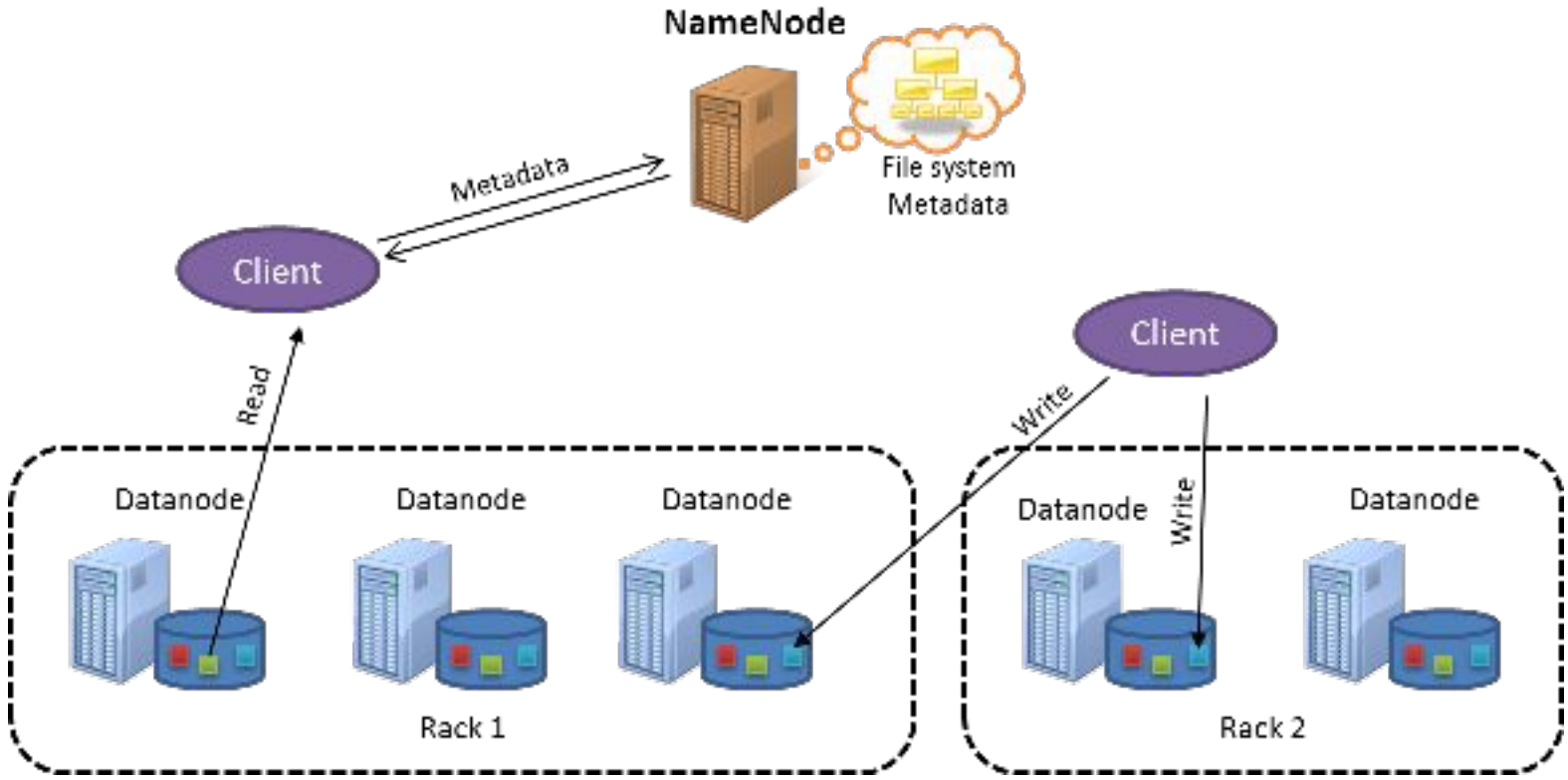


HDFS - Distributed File System

- Paper
  - The Hadoop Distributed File System, Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, Yahoo!, 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)
- Purpose
  - Serve as the distributed storage to run Hadoop's MapReduce applications
  - An open-source framework which can be used by different clients with different needs

HDFS - Distributed File System

- Hadoop Distributed File System
- Open source version of Google File System

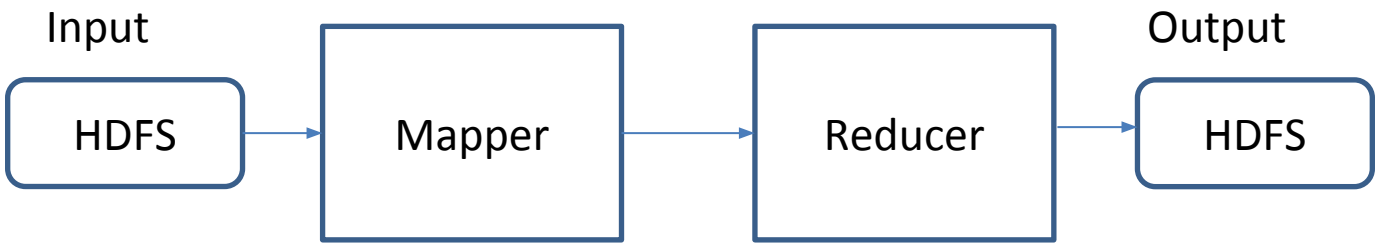


Using a Custom Jar in P4.1

- What is a custom JAR
  - Customize your java MapReduce program
  - Run the MapReduce JAR in EMR
- Why custom JAR
  - More resources: HDFS/HBASE/S3
  - More job configuration flexibility
  - More control of how the resources are utilized

Typical MapReduce Job

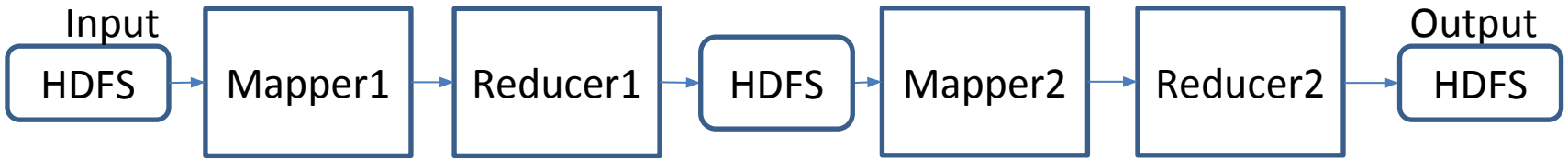
- Simplistic view of a MapReduce job



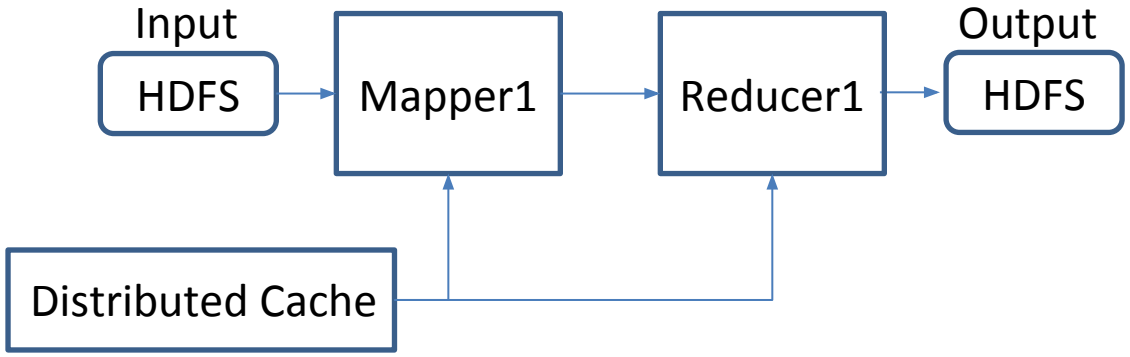
- You simply write code for the
  - Mapper
  - Reducer

Cool things with MapReduce

- Chain of two MapReduce jobs



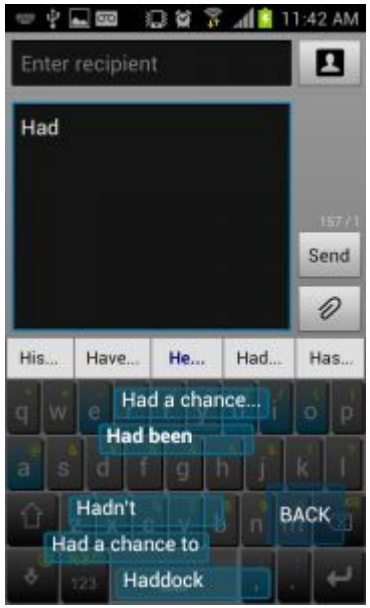
- Load external data into your program

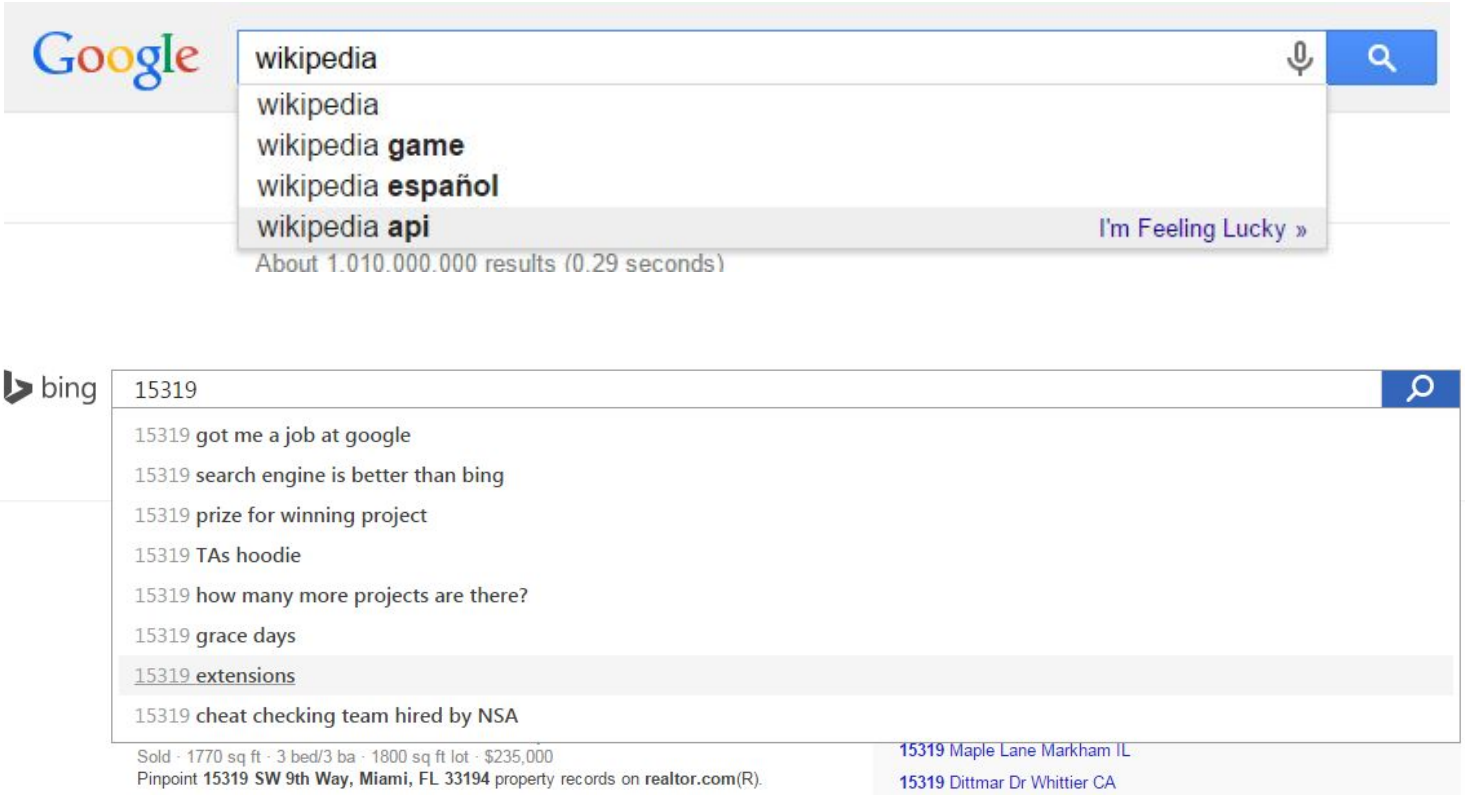


- Modify the behavior of FileInputSplit
- Load output of MapReduce job to HBase

Project 4.1 - Input Text Predictor

- Suggest words based on phrases already typed





Project 4.1

- Input Text Predictor
  - Input Data
  - N-Gram Model
  - Statistical Language Model
  - Predict the next word given a phrase
- Have to use EMR Custom JAR
  - **CANNOT use EMR Streaming**

Construct an Input Text Predictor

1. Given a language corpus
   - Wikipedia dataset (~4.5 GB)
2. Construct an n-gram model of the corpus
   - An n-gram is a phrase with n contiguous words
   - For example a set of 1,2,3,4,5-grams with counts:
     - this 1000
     - this is 500
     - this is a 125
     - this is a cloud 60
     - this is a cloud computing 20

Construct an Input Text Predictor - 2

3. Build a statistical language model to calculate the probability of a word appearing after a phrase

$$\Pr(\text{word} \mid \text{phrase}) = \frac{\text{Count}(\text{phrase} + \text{word})}{\text{Count}(\text{phrase})}$$

$$\Pr(\text{is} \mid \text{this}) = \frac{\text{Count}(\text{this is})}{\text{Count}(\text{this})} = \frac{500}{1000} = 0.5$$

$$\Pr(a \mid \text{this is}) = \frac{\text{Count}(\text{this is } a)}{\text{Count}(\text{this is})} = \frac{125}{500} = 0.25$$

4. Load data to HBase and predict the next word based on the probabilities

Generate n-gram

- An *n*-gram is a phrase with *n* contiguous words

Example Phrase: This is interesting because this is a cloud computing course

#	1-gram	Count
1	this	2
2	is	2
3	interesting	1
4	because	1
5	a	1
6	cloud	1
7	computing	1
8	course	1

#	2-gram	Count
1	this is	2
2	is interesting	1
3	interesting because	1
4	because this	1
5	is a	1
6	a cloud	1
7	cloud computing	1
8	computing course	1

#	3-gram	Count
1	this is interesting	1
2	is interesting because	1
3	interesting because this	1
4	because this is	1
5	this is a	1
6	is a cloud	1
7	a cloud computing	1
8	cloud computing course	1

#	4-gram	Count
1	this is interesting because	1
2	is interesting because this	1
3	interesting because this is	1
4	because this is a	1
5	this is a cloud	1
6	is a cloud computing	1
7	a cloud computing course	1
8		

#	5-gram	Count
1	this is interesting because this	1
2	is interesting because this is	1
3	interesting because this is a	1
4	because this is a cloud	1
5	this is a cloud computing	1
6	is a cloud computing course	1
7		
8		

#	6-gram	Count
1	this is interesting because this is	1
2	is interesting because this is a	1
3	interesting because this is a cloud	1
4	because this is a cloud computing	1
5	this is a cloud computing course	1
6		
7		
8		

Statistical Language Model

- Provide a mechanism to solve common natural language processing problems
- Examples: speech recognition, machine translation and intelligent input method
- SLM estimates the probability of a word given the previous phrase
- N-gram model is one of the most popular mechanisms to generate an SLM today

Statistical Language Model

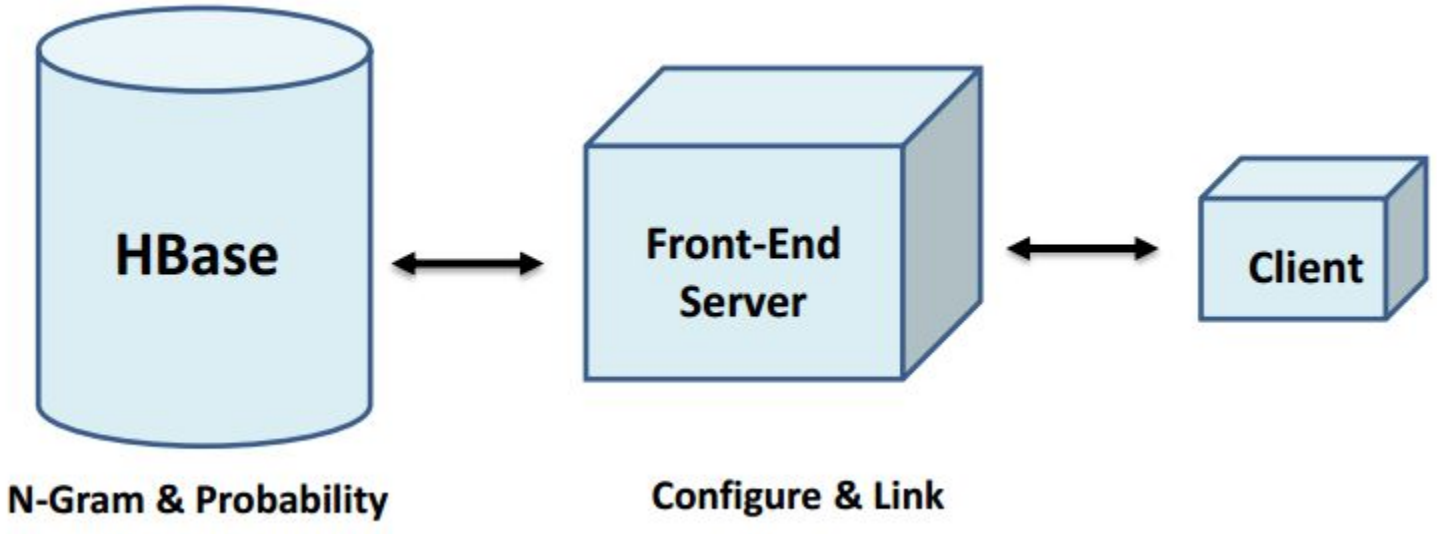
- Build a statistical language model that calculates the probability of a word appearing after a phrase

Options	Count	Probability
this was	150	0.15
this is	500	0.50
this day	250	0.25
this kiss	25	0.03
this boy	75	0.08

Options	Count	Probability
this is	500	0.50
this day	250	0.25
this was	150	0.15
this boy	75	0.08
this kiss	25	0.03

Load and Predict

- Load data into HBase
- Connect HBase with the PHP-based front end server to provide a functional web service.



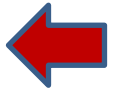
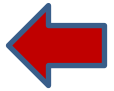
Recommendations

- Test for correctness with small datasets first
- **Don't** start a new cluster for every job
  - EMR will charge you one hour of usage for instances even though your EMR job failed to start
- Optimize your code to accelerate MapReduce before seeking other optimization methods
  - Pay attention to your code efficiency
- Version of Hadoop
  - should match the version shown in EMR AMI
- Start early

Module to Read

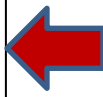
- UNIT 5: Distributed Programming and Analytics Engines for the Cloud
  - Module 18: Intro to distributed programming for the Cloud
  - Module 19: Distributed analytics engines: MapReduce
  - Module 20: Distributed analytics engines: Spark
  - Module 21: Distributed analytics engines: GraphLab

Upcoming Deadlines

- Quiz 11 : Unit 5 - Module 19 
  - Open: 11/20/2015 12:01 AM Pittsburgh
  - Due: 11/20/2015 11:59 PM Pittsburgh
- Project 4.1 : Batch Processing with MapReduce 
  - Due: 11/22/2015 11:59 PM Pittsburgh
- 15619Project : Phase 3
  - Live-test due: 12/02/2015 4:59 PM Pittsburgh
  - Code and report due: 12/03/2015 11:59 PM Pittsburgh

Busy Week Coming Up!

Wednesday	Thursday	Friday	Sunday
Wednesday 12/2/2015 20:00:01 EST • Phase 3 Live Test	Thursday 12/3/2015 23:59:59 EST • Phase 3 Code & Report Due	Friday 12/4/2015 23:59:59 EST • Quiz 12	Sunday 12/6/2015 23:59:59 EST • P4.2 Due



Project 4.1

- Steps and Demo

Overview

- Run 2 MapReduce jobs to generate a language model
- First step: generate n-grams
- Second step: generate language model
- Third step: connect user interface to HBase

Grading, Step 1

- Use submitter file to autograde your answers
- Ngrams
- Run the command *./ngram_submitter* with the top 100 ngrams in a file called “ngrams”.
- Important Note: Make sure you create a submission folder. Do not run the submitter from the home folder.

Grading, Steps 2 & 3

- Model
- Run the command *./submitter* to autograde you language model and your interface
- Code files for ngram and model will be manually graded

Bonus

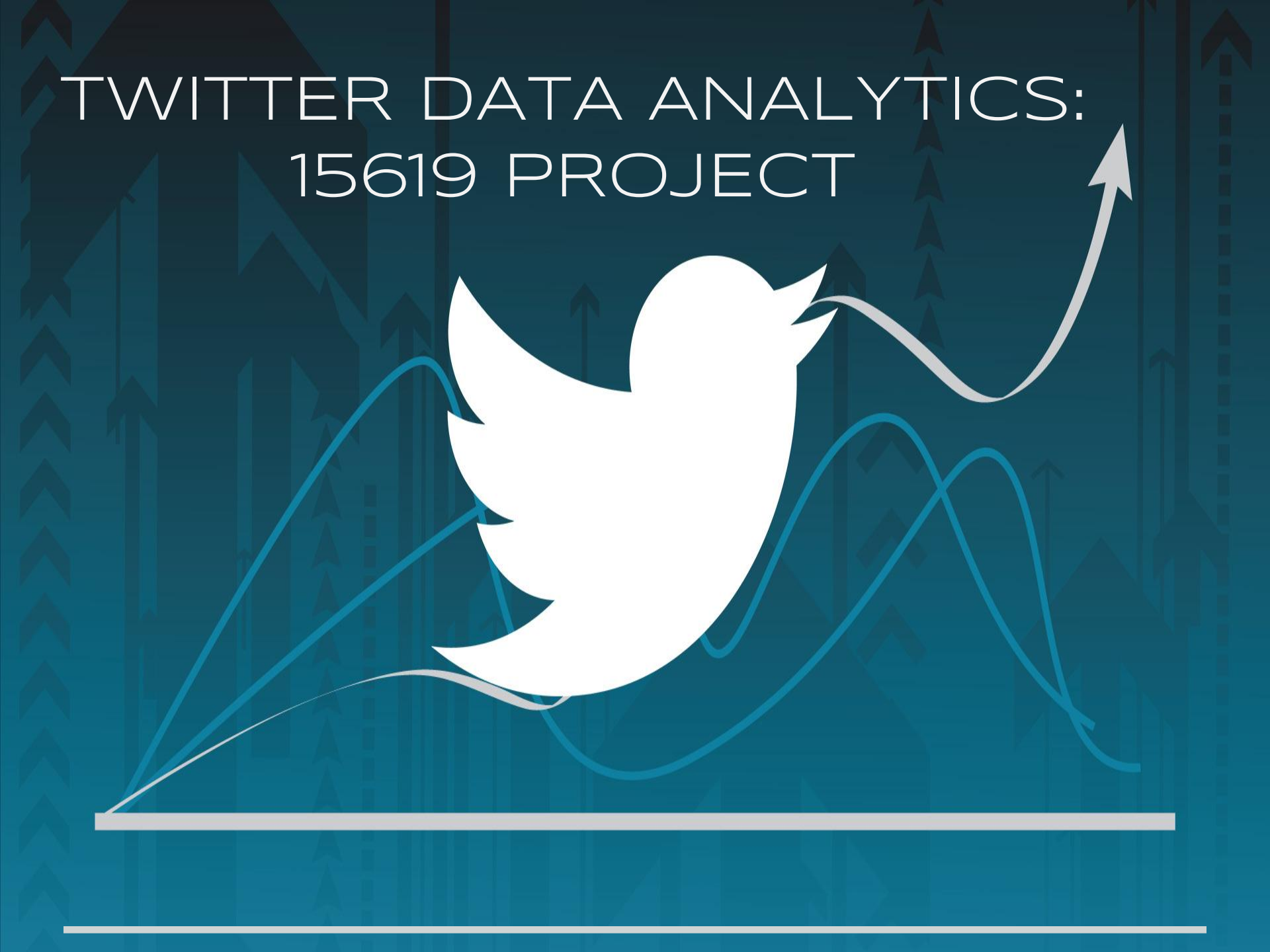
- Extending the ideas to build a character-gram model for word auto-completion
- Simple word-count will suffice
- Use the bonus_submitter
- 20% of P4.1 grade

Hive Shell for ad-hoc queries

- Run SQL like queries over distributed storage(HDFS/S3)
- SELECT, WHERE, ORDER BY,...
- https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select

Questions?

TWITTER DATA ANALYTICS: 15619 PROJECT



15619Project Agenda

- Phase 2 Scores
- Query 5 Discussion
- Query 6 Discussion
- Upcoming Deadlines

Phase 2 Live Test

Congratulations to the Teams on the Leaderboard!

MySQL

RPComputing	100
Omegaga's_Black_Railgun	100
AmazonCEO	100
Clove	99.54
purrito	98.5
hakunamatata	97.39
Funny	96.16
CCCoder	94.34
Night Run	93.66
Yamete	93.25

HBase:

QingfenSpicyPot	99.78
CloudBurst1	99.61
CCCoder	98.44
i2.8xlarge	64.55
carpeDiem	60.6
theImp	52.91
Yolo	52.78
purrito	52.21
RPComputing	51.24
Omegaga's_Black_Railgun	50.55

Combined Top 10 (includes bonuses)

Team	HBase	MySQL	Phase 2 Final
CCCoder	98.44	94.34	109.69
CloudBurst1	99.61	93.1	109.56
QingfenSpicyPot	99.78	91.7	108.84
i2.8xlarge	64.55	89.47	85.31
RPComputing	51.24	100	82.92
purrito	52.21	98.5	82.66
Omegaga's_Black_Railgun	50.55	100	82.48
carpeDiem	60.6	81.17	77.89
ace619	45.44	92.5	70.97
qitiandasheng	45.43	89.78	69.61

Query 5: Tweet Counter

- **Description:** The query asks for the total number of tweets sent by all users given a range of userids.
- **Request:** We send you two user ids

```
GET /q5?userid_min=u_id&userid_max=u_id
```

- **Response:** Your web service needs to return the number of tweets sent within the range of user ids where user ids are inclusive
- **Warning:** Ignore duplicate tweet IDs (Count once)

Query 5: Tweet Counter

```
GET /q5?userid_min=2&userid_max=10
```

User ID	Tweet ID
1	101
1	102
2	103
2	104
3	105
3	105
4	106
7	107
10	108

Response Format:

```
TEAMID,TEAM_AWS_ACCOUNT_IDs\n
Count\n
```

Guess the Response:

```
TEAMID,TEAM_AWS_ACCOUNT_IDs\n
6\n
```

Query 6: Tweet Tagger

Finally, we're dealing with writes!!!

- Append a random string to the end of an existing tweet
- **Each tweet can have only a single appended tag at a time** (last writer wins)
- ETL similar to /q2 (with no date limits)
- When we read in Q6, we expect to see the censored tweet text, with appended tag (if any)
- Correctness test strictest for reads

Query 6: Tweet Tagger

- Problem: Request Reordering
  - The old LG scheme cannot guarantee that you will receive and apply operations in the order that we expect
  - Example: Network delay causes a read to happen before a write, which was not expected by the grader

- Solution: Sequence Numbers
  - All reads (opt = r) and appends (opt = a) have a sequence number (seq = 1 to 5)
  - ALL Traffic divided into transactions of length = 5

Query 6: Tweet Tagger

- Designing a replicated backend
  - Ideally, ensure that a write updates all replicas before reading from any replica
  - Faster: Only read from the most “recently updated replica”
  - Or: Update all replicas asynchronously (for ideas, see chain replication, other schemes in Ceph)
  - Tradeoffs: Accuracy v/s Performance

Query 6: Tweet Tagger

- Designing a sharded backend
  - Split data between nodes based on keys
  - Benefit: More space/memory efficient

  - If using ELB:
    - Your front-end may need to be node-aware
    - Extra hop?

  - If not using ELB:
    - Consider nginx or HAProxy or other LBs

- How well you distribute data is crucial

Query 6: Tweet Tagger

Consider Tweet ID: **44898831041785037O**

@Maria_LeonPL chulada de mujeres....sensacional paisana... estaremos atento de su intervención... besos tu caballero de la noche¡¡

Query 6: Tweet Tagger

- Step 1 : Start transaction (**opt=s**)

```
/q6?opt=s&tid=3000001

TEAMID,TEAM_AWS_ACCOUNT_ID\n

0\n
```

- Hint:
  - All transactions operate on an independent set of tweet IDs

Query 6: Tweet Tagger

- Step 2 : Exactly 5 Appends (**opt=a**) or Reads (**opt=r**)

```
/q6?
tid=3000001&seq=1&opt=a&tweetid=4489883104178503
70&tag=ILOVE15619!12
TEAMID,TEAM_AWS_ACCOUNT_ID\n
ILOVE15619!12\n
```

- Hint:
  - When opt=a, return the tag to the user
  - Scope for optimization? Yes, but be careful!!!

Query 6: Tweet Tagger

- Step 2 : Exactly 5 Appends (**opt=a**) or Reads (**opt=r**)

```
/q6?
tid=3000001&seq=2&opt=r&tweetid=448988310417850370
TEAMID,TEAM_AWS_ACCOUNT_ID\n
@Maria_LeonPL chulada de mujeres....sensacional
paisana...estaremos atento de su intervención...
besos tu caballero de la noche¡¡ILOVE15619!12\n
```

Query 6: Tweet Tagger

- Step 2 : Exactly 5 Appends (**opt=a**) or Reads (**opt=r**)

```
/q6?
tid=3000001&seq=4&opt=a&tweetid=448988310417850371&tag=ILOVE15619!13
TEAMID,TEAM_AWS_ACCOUNT_ID\n
ILOVE15619!13\n
```

- Note:
  - If you receive an operation out of order, you need to ensure that the previous operation is performed first
  - Multiple tweet IDs may be operated on in a single transaction

Query 6: Tweet Tagger

- Step 2 : Exactly 5 Appends (**opt=a**) or Reads (**opt=r**)

```
/q6?
tid=3000001&seq=3&opt=a&tweetid=448988310417850371&tag=ILOVE15619!14
TEAMID,TEAM_AWS_ACCOUNT_ID\n
ILOVE15619!14\n
```

- Note:
  - If you receive an operation out of order, you need to ensure that the previous operation is performed first
  - Multiple tweet IDs may be operated on in a single transaction

Query 6: Tweet Tagger

- Step 2 : Exactly 5 Appends (**opt=a**) or Reads (**opt=r**)

```
/q6?
tid=3000001&seq=5&opt=r&tweetid=448988310417850370
TEAMID,TEAM_AWS_ACCOUNT_ID\n
@Maria_LeonPL chulada de mujeres....sensacional paisana...estaremos atento de su intervención... besos tu caballero de la noche¡¡ILOVE15619!12\n
```

Query 6: Tweet Tagger

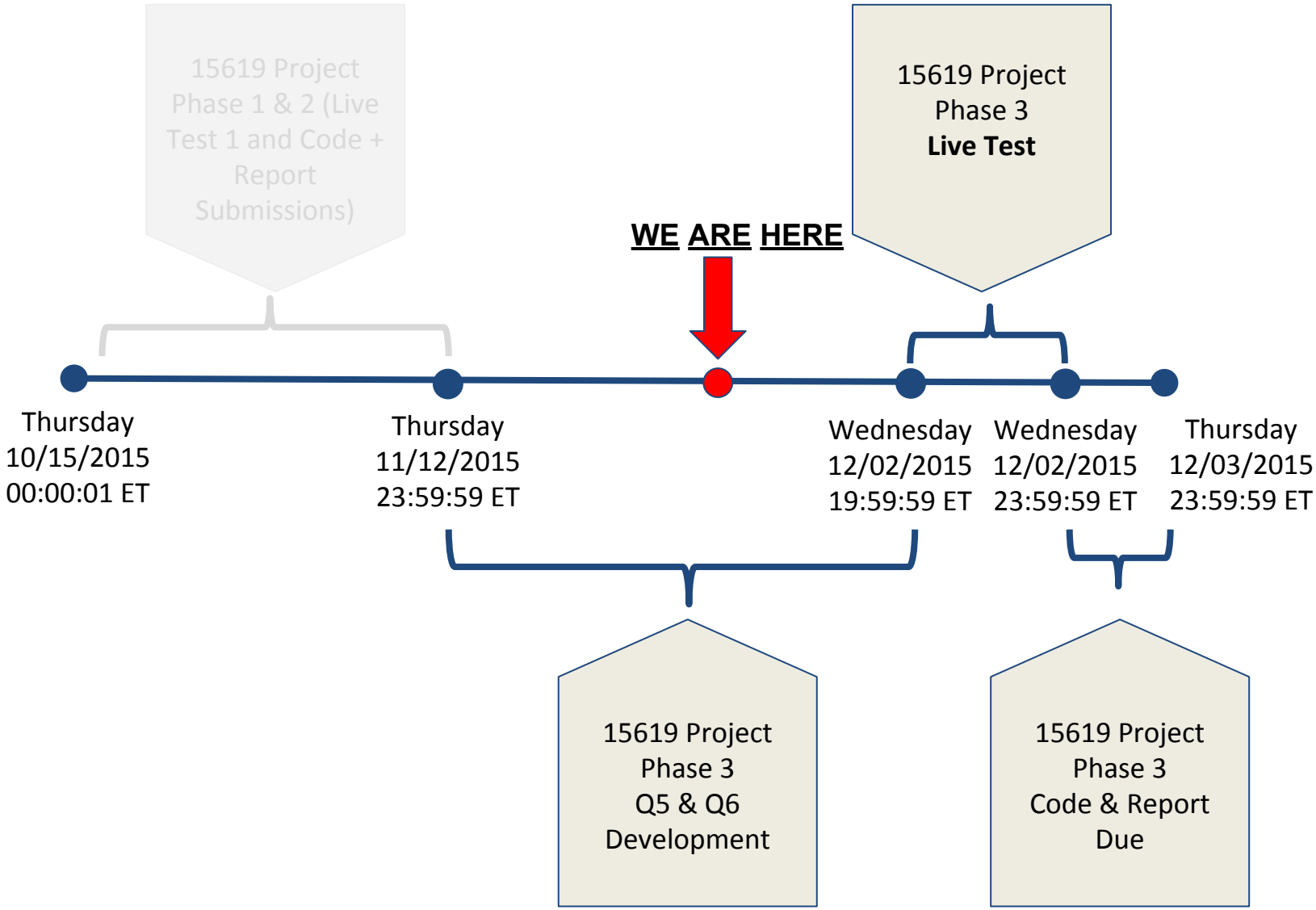
- Step 3 : End Transaction (**opt=e**) or Reads (**opt=r**)

```
/q6?tid=3000001&opt=e

TEAMID,TEAM_AWS_ACCOUNT_ID\n
0\n
```

- Note:
  - Multiple simultaneous, overlapping transactions
  - Ensure that all 5 sequence numbers are handled

15619 Project Phase 3 Deadlines



What's due next?

- Phase 3 Deadline
  - **Submission of one URL by 18:59 ET (Pittsburgh) Wed 12/2**
    - **Live Test from 8 PM to midnight ET**
  - Choose any one (or both) databases
  - Can only use m1.large or cheaper t1, t2, m1, m3 instances
  - Fix Q1, Q2, Q3, Q4 if your Phase 2 did not go well
  - New queries Q5 and Q6.
  - Phase 3 counts for **60%** of the 15619Project grade

15619Project Phase 3 Live Test

- 30 minutes warm-up (Q1 only)
- 3 hours Q1 - Q6
- 30 minutes mix-Q1+Q2+Q3+Q4+Q5
- Preparing for the live test
  - Choose a database based on your observations from previous phases and all six queries
  - Caching known requests will not work(unless you are smart)
  - Need to have all Qs running at the same time
  - Avoid bottlenecks in mixed queries