15-319 / 15-619
Cloud Computing

Recitation 12
April 7th and 9th, 2015
Overview

• Last week’s reflection
  – Project 3.5
• Budget issues
  – Tagging, 15619Project
• This week’s schedule
  – Unit 5 - Modules 17
  – Project 4.1
• Demo
• Twitter Analytics: The 15619Project
Reflections on P3.5

• Implement a key-value store with different levels of consistency
  • Strong consistency
  • Causal consistency

• FAQs
  • Failure in consistency checker test
    • Best way: log requests and analyze
    • The checker also checks for response time of certain requests in some tests
Budget Issues

- Tag all resources (Piazza @1656)
  - Amazon EBS General Purpose SSD
  - Amazon EBS Provisioned IOPS
  - Amazon EBS Snapshot
- Untagged resources will be counted towards your weekly project
  - Keep a buffer
Module to Read

• UNIT 5: Distributed Programming and Analytics Engines for the Cloud
  – Module 16: Introduction to Distributed Programming for the Cloud
  – Module 17: Distributed Analytics Engines for the Cloud: MapReduce
    • Hadoop 1.0
    • Hadoop 2.0 - YARN
  – Module 18: Distributed Analytics Engines for the Cloud: Spark
  – Module 19: Distributed Analytics Engines for the Cloud: GraphLab
Project 4

• Project 4.1
  – MapReduce Programming Using YARN

• Project 4.2
  – Iterative Programming Using Apache Spark

• Project 4.3
  – Graph Programming Using GraphLab
MapReduce - Introduced in Project 1

• The idea of MapReduce

Apple, 1
Apple, 1
Apple, 1
How many times does the word “Apple” appear in these books?

I heard 6 “Apple”s!
MapReduce - Overview

- The idea of MapReduce

How do I know who is the “Apple” listener?

You Don’t!
MapReduce - Phases

• The idea of MapReduce

Map Phase

- Orange, 1
- Blueberry, 1
- Blueberry, 1
- Apple, 1

- Apple, 1
- Apple, 1
- Apple, 1
- Orange, 1

- Apple, 1
- Apple, 1
- Orange, 1
- Blueberry, 1

Reducer

Magic Box (Shuffle, sort, merge)

- Orange ?
- Apple ?
- Blueberry ?

Map Phase

- Mapper

Reducer
• The idea of MapReduce

Map Phase

Reduce Phase

Magic Box (Shuffle, sort, merge)
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

\[(k2, \text{list}(v2)) \rightarrow \text{Reducer} \rightarrow (k3, v3)\]
Proprietary

Google

MapReduce

BigTable

Open Source

hadoop

HDFS

MapReduce

HBase
Hadoop

• MapReduce
  – A programming model for processing large data sets using a parallel distributed algorithm

• Apache Hadoop
  – A framework for running 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
HDFS - Distributed File System

• Paper

• 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
MapReduce and HDFS

- Detailed workflow
Cool things with MapReduce

- Chain of two MapReduce jobs

- Load external data into your program

- Modify the behavior of FileInputSplit
Project 4.1 - Input Text Predictor

- Suggest words based on letters 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
   – Project Gutenberg (2.5 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

\[
\Pr(\text{word} | \text{phrase}) = \frac{\text{Count(phrase + word)}}{\text{Count(phrase)}}
\]

\[
\Pr(\text{is} | \text{this}) = \frac{\text{Count(\text{this} is)}}{\text{Count(\text{this})}} = \frac{500}{1000} = 0.5
\]

\[
\Pr(\text{a} | \text{this is}) = \frac{\text{Count(\text{this is a})}}{\text{Count(\text{this is})}} = \frac{125}{500} = 0.25
\]

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

<table>
<thead>
<tr>
<th>#</th>
<th>1-gram</th>
<th>Count</th>
<th>2-gram</th>
<th>Count</th>
<th>3-gram</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>this</td>
<td>2</td>
<td>this is</td>
<td>2</td>
<td>this is interesting</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>is</td>
<td>2</td>
<td>is interesting</td>
<td>1</td>
<td>is interesting because</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>interesting</td>
<td>1</td>
<td>interesting because</td>
<td>1</td>
<td>interesting because this</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>because</td>
<td>1</td>
<td>because this</td>
<td>1</td>
<td>because this is</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>1</td>
<td>is a</td>
<td>1</td>
<td>this is a</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>cloud</td>
<td>1</td>
<td>a cloud</td>
<td>1</td>
<td>is a cloud</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>computing</td>
<td>1</td>
<td>cloud computing</td>
<td>1</td>
<td>a cloud computing</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>course</td>
<td>1</td>
<td>computing course</td>
<td>1</td>
<td>cloud computing course</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>4-gram</th>
<th>Count</th>
<th>5-gram</th>
<th>Count</th>
<th>6-gram</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>this is interesting</td>
<td>1</td>
<td>this is interesting because this</td>
<td>1</td>
<td>this is interesting because this is</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>is interesting</td>
<td>1</td>
<td>is interesting because this is</td>
<td>1</td>
<td>is interesting because this is a</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>interesting</td>
<td>1</td>
<td>interesting because this is a</td>
<td>1</td>
<td>interesting because this is a cloud</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>because this is a</td>
<td>1</td>
<td>because this is a cloud</td>
<td>1</td>
<td>because this is a cloud computing</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>this is a cloud</td>
<td>1</td>
<td>this is a cloud computing</td>
<td>1</td>
<td>this is a cloud computing course</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>is a cloud computing</td>
<td>1</td>
<td>is a cloud computing course</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>a cloud computing</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>cloud computing course</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.
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
Upcoming Deadlines

● P4.1 MapReduce Programming Using YARN
  ○ Due: 11:59PM ET April 12th (Sunday)

● 15619 Project Phase 3
  ○ Deadline: 18:59PM ET April 15th (Wednesday)
    ■ Live Test, due 18:59PM ET Apr 15th
Project 4.1

- 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

● Use submitter file to autograde your answers
● Ngrams
● Run the command 
  
  ./submitter -n with the top
  
  100 ngrams in a file called “ngrams”.
Grading

- Model
- Run the command `./submitter -m` to autograde your language model and your interface
- Code files for ngram and model will be manually graded
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](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select)
Questions?
TWITTER ANALYTICS: THE 15619PROJECT
Phase 2 Live Test

Congratulations UnusualItem!

MySQL

<table>
<thead>
<tr>
<th>Team</th>
<th>Phase Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnusualItem</td>
<td>203</td>
</tr>
<tr>
<td>ComeATeamName</td>
<td>186</td>
</tr>
<tr>
<td>HYPER</td>
<td>160</td>
</tr>
<tr>
<td>CloudWalker</td>
<td>149</td>
</tr>
<tr>
<td>z2m</td>
<td>99</td>
</tr>
<tr>
<td>T.K.Lim</td>
<td>95</td>
</tr>
<tr>
<td>Lays</td>
<td>92</td>
</tr>
<tr>
<td>Oak</td>
<td>89</td>
</tr>
<tr>
<td>Penguins</td>
<td>89</td>
</tr>
<tr>
<td>YouKnowWho</td>
<td>86</td>
</tr>
<tr>
<td>etc</td>
<td>75</td>
</tr>
</tbody>
</table>

HBase

<table>
<thead>
<tr>
<th>Team</th>
<th>Phase Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnusualItem</td>
<td>90</td>
</tr>
<tr>
<td>T.K.Lim</td>
<td>82</td>
</tr>
<tr>
<td>YouKnowWho</td>
<td>68</td>
</tr>
<tr>
<td>The Three Foes</td>
<td>57</td>
</tr>
<tr>
<td>Thunder &amp; Lightning</td>
<td>57</td>
</tr>
<tr>
<td>ComeATeamName</td>
<td>49</td>
</tr>
<tr>
<td>Wegotateam</td>
<td>46</td>
</tr>
<tr>
<td>Oak</td>
<td>44</td>
</tr>
<tr>
<td>etc</td>
<td>43</td>
</tr>
<tr>
<td>Cloud007</td>
<td>41</td>
</tr>
<tr>
<td>9527</td>
<td>41</td>
</tr>
</tbody>
</table>
What’s due soon?

● Phase 3 Deadline
  ○ Submission of one URL by 18:59 ET (Pittsburgh) Wed 4/15
    ■ Live Test from 8 PM to midnight ET
  ○ Choose one database
  ○ 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 15619Project grade
Phase 3 Report [VERY IMPORTANT]

- Start early
- Document your steps
- Identify and isolate the performance impact of each change you make
- Document your ideas and experiments

MAKE A QUANTITATIVE, DATA-DRIVEN REPORT
Query 5: Twitter Rankster

- Request: a list of userids and a date range
- You should award points to the users based on these rules:
  - +1 per unique tweet sent by the user in the time interval
  - +3 per friend (based on the maximum value of user.friends_count in the time interval)
  - +5 per follower (based on the maximum value of user.followers_count in the time interval)
Query 5: Twitter Rankster

GET /q5?userlist=12,14,16,18,20&start=2010-01-01&end=2014-12-31

Team,1234-5678-1234,1234-5678-1234,1234-5678-1234
12,173
16,155
14,99
20,99
18,55
Query 6: Hermit Finder

- Request: A range of userids
- You should count the number of users where:
  - userid is between M and N inclusive,
  - has at least one tweet but none of his/her tweets contain location information.

GET /q6?m=0&n=9999999999

Team,1234-5678-1234,1234-5678-1234,1234-5678-1234
55811730
Live Test

- 30 minutes warm-up
- 3 hours Q1 - Q6
- 30 minutes mix-Q1Q2Q3Q4Q5Q6
- 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
  - Don’t expect testing in sequence
  - Avoid bottlenecks in mixed queries
Warnings

● Avoid tagging penalties

● Keep a watch on budget.
  ○ $55 (phase + livetest)

● Check correctness before livetest

● Start early