Overview

• Last week’s reflection
  – 15619Project Winners
• Fall 2015 TA Application
• This week’s schedule
  – Unit 5 - Module 19
  – Project 4.2
  – Quiz 5
• Demo
15619Project - Congratulations!

For Q1: Throughput Constraint 15,000

<table>
<thead>
<tr>
<th>Team</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyper</td>
<td>21,405</td>
</tr>
<tr>
<td>Cloud007</td>
<td>21,177</td>
</tr>
<tr>
<td>Lays</td>
<td>21,144</td>
</tr>
<tr>
<td>abracadabra</td>
<td>20,948</td>
</tr>
<tr>
<td>9527</td>
<td>20,897</td>
</tr>
</tbody>
</table>
**15619Project - Congratulations!**

**For Q2: Throughput Constraint 6,000**

<table>
<thead>
<tr>
<th>Team</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opener</td>
<td>17,165</td>
</tr>
<tr>
<td>MadHatters</td>
<td>17,050</td>
</tr>
<tr>
<td>Orions</td>
<td>16,961</td>
</tr>
<tr>
<td>--</td>
<td>16,239</td>
</tr>
<tr>
<td>Kaale Baadal</td>
<td>16,204</td>
</tr>
</tbody>
</table>
15619Project - Congratulations!

For Q3: Throughput Constraint 10,000

<table>
<thead>
<tr>
<th>Team</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>HYPER</td>
<td>16,949</td>
</tr>
<tr>
<td>AataMaazhiSatakli</td>
<td>16,947</td>
</tr>
<tr>
<td>abracadabra</td>
<td>16,598</td>
</tr>
<tr>
<td>T.K.Lim</td>
<td>16,561</td>
</tr>
<tr>
<td>Penguins</td>
<td>16,496</td>
</tr>
</tbody>
</table>
15619 Project - Congratulations!

For Q4: Throughput Constraint 6,000

<table>
<thead>
<tr>
<th>Team</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnusualItem</td>
<td>12,453</td>
</tr>
<tr>
<td>Cloud Ninjas</td>
<td>12,422</td>
</tr>
<tr>
<td>Lays</td>
<td>12,315</td>
</tr>
<tr>
<td>T.K.Lim</td>
<td>12,319</td>
</tr>
<tr>
<td>abracadabra</td>
<td>12,269</td>
</tr>
</tbody>
</table>
For Q5: Throughput Constraint 4,000

<table>
<thead>
<tr>
<th>Team</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orion</td>
<td>18,369</td>
</tr>
<tr>
<td>UnderTheCloud</td>
<td>17,547</td>
</tr>
<tr>
<td>Cloud Ninjas</td>
<td>15,104</td>
</tr>
<tr>
<td>UnusualItem</td>
<td>14,816</td>
</tr>
<tr>
<td>T.K.Lim</td>
<td>14,787</td>
</tr>
</tbody>
</table>
# 15619Project - Congratulations!

For Q6: Throughput Constraint 10,000

<table>
<thead>
<tr>
<th>Team</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnusualItem</td>
<td>19,516</td>
</tr>
<tr>
<td>z2m</td>
<td>18,307</td>
</tr>
<tr>
<td>YouKnowWho</td>
<td>17,032</td>
</tr>
<tr>
<td>Cloud Ninjas</td>
<td>17,001</td>
</tr>
<tr>
<td>CloudWalker</td>
<td>16,480</td>
</tr>
</tbody>
</table>
15619Project - Congratulations!

Overall: Maximum Score 100

<table>
<thead>
<tr>
<th>Rank</th>
<th>Team</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T.K.Lim</td>
<td>225</td>
</tr>
<tr>
<td>2</td>
<td>9527</td>
<td>194</td>
</tr>
<tr>
<td>3</td>
<td>YouKnowWho</td>
<td>193</td>
</tr>
<tr>
<td>4</td>
<td>Cloud Ninjas</td>
<td>167</td>
</tr>
<tr>
<td>5</td>
<td>z2m</td>
<td>143</td>
</tr>
<tr>
<td>6</td>
<td>Penguins</td>
<td>99</td>
</tr>
<tr>
<td>7</td>
<td>THU</td>
<td>99</td>
</tr>
<tr>
<td>8</td>
<td>UnusualItem</td>
<td>98</td>
</tr>
<tr>
<td>9</td>
<td>HereComesTheHotTepper</td>
<td>98</td>
</tr>
<tr>
<td>10</td>
<td>Thunder &amp; Lightning</td>
<td>97</td>
</tr>
</tbody>
</table>

T.K Lim Team:  
- Yuxuan Wan  
- Yiming Zang  
- Yishuang Lu

Winning Team!!!

Winning Team:
- Yuxuan Wan  
- Yiming Zang  
- Yishuang Lu
Fall 2015 TA Applications

● Apply here or see Piazza for more...

● Learn the Dark Arts
  ○ See how the MSB works
  ○ Upgrade The Project Zone to
    ■ allow content, activities, quizzes
    ■ become a comprehensive learning portal
    ■ support more courses
  ○ Extend the 15619Project (scale, operations)
  ○ Create new project modules (P1/2/3/4)
  ○ Improve project writeups, feedback
  ○ Write content from state-of-the-art papers
  ○ Design “Minority Report”-like anti-cheating
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
  – 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
Typical MapReduce Job

- Simplistic view of a MapReduce job
  - You simply write code for the
    - Mapper
    - Reducer
  - Inputs are read from disk and outputs are written to disk
    - Intermediate data is spilled to local disk
Iterative MapReduce Jobs

• Some applications require iterative processing
• Eg: Machine Learning, etc.

- MapReduce: Data is always spilled to disk
  - Added overhead for each iteration
  - Can we keep data in memory? Across Iterations?
  - How do you manage this?
The Spark Framework

 RDD Objects

 Spark Client (Application Master)

 Task Scheduler

 Worker

 rdd1.join(rdd2).
groupBy(...).
.filter(...)

 Scheduler and RDD Graph

 Cluster Manager

 Threads

 Block Manager

 BlockInfo

 MemoryStore

 DiskStore

 ShuffleBlockManager
Resilient Distributed Datasets (RDDs)

- RDDs are
  - in-memory
  - read-only objects
  - partitioned across the cluster
  - partitioned across machines based on a range or the hash of a key in each record
Operations on RDDs

- **Loading**
  - `>>>input_RDD = sc.textFile("text.file")`

- **Transformation**
  - Apply an operation and derive a new RDD
  - `>>>transform_RDD = input_RDD.filter(lambda x: "abcd" in x)`

- **Action**
  - Computations on an RDD that return a single object
  - `>>>print "Number of abcd:" + transform_RDD.count()`
RDDs and Fault Tolerance

- Actions create new RDDs
- Instead of replication, recreate RDDs on failure
- Use RDD lineage
  - RDDs store the transformations required to bring them to current state
  - Provides a form of resilience even though they are in-memory
Spark Ecosystem

- **Spark SQL**
  – Allows running of SQL-like queries against RDDs

- **Spark Streaming**
  – Run spark jobs against streaming data

- **MLlib**
  – Machine learning library

- **GraphX**
  – Graph-parallel framework
Project 4.2

• Build a basic search engine!
  – Index pages
  – Run PageRank

• No need to include an input text predictor
  – Unless you want to and have time and budget to spare
Project 4.2 - Overview

- Use the Wikipedia English dataset
- Build a search engine for Wikipedia documents
- Find and order the pages using
  - Term Frequency - Inverse Document Frequency (TF-IDF)
  - PageRank
Project 4.2 - Two Parts

1. Generate TF-IDF
   – Get weighted measurements of individual words in the document
2. Run PageRank
   – Rank pages according to the incoming and outgoing links
TF-IDF

• Term Frequency (TF)
  – The count of the number of times a word has appeared in a document

• Inverse Document Frequency (IDF)
  – The total number of documents / number of documents in which a particular word has appeared

• For a term $i$ in document $j$

\[
    w_{i,j} = tf_{i,j} \times \log \left( \frac{N}{df_i} \right)
\]

- $tf_{i,j} = \text{number of occurrences of } i \text{ in } j$
- $df_i = \text{number of documents containing } i$
- $N = \text{total number of documents}$
PageRank

• Give pages ranks (scores) based on links to them
• A page that has:
  – Links from many pages ⇒ high rank
  – Link from a high-ranking page ⇒ high rank
PageRank

For each Page i in dataset, Rank of i can be computed:

\[ R[i] = 0.15 + \sum_{j \in \text{Nbr}(i)} w_{ji} \times R[j] \]

Iterate until R[i] converges

Formula to be implemented for 4.2 is slightly more complex. Read carefully!!!
Launching a Spark Cluster

- Use the Spark-EC2 scripts
- Command line options to specify instance types and spot pricing
- Spark is an in-memory system
  - r3 EC2 instances are a good match as they have a lot of RAM
  - test with a single instance first
- Develop and test your scripts on a portion of the dataset before launching a cluster
Spark Shell

• Like the python shell

• Run commands interactively

• On the master, execute (from /root)
  – ./spark/bin/spark-shell
  – ./spark/bin/pyspark
Grading

• Use submitter file to autograde your answers

• For TF-IDF
  o Run the command `./submitter -t`
    ▪ Uploads code immediately
    ▪ Instant feedback (like Project 4.1)

• For PageRank
  o Run the command `./submitter -p`
    ▪ Uploads code immediately
    ▪ Instant feedback (like Project 4.1)
Quiz 5

- Quiz 5 will be open for 24 hours, Friday, Apr 24
  - Quiz 5 becomes available on Apr 24, 00:01 AM EST.
  - Deadline for submission is Apr 24, 11:59 PM EST.
  - Once open, you have **180 min** to complete the quiz.
  - Late submissions are NOT accepted.
  - You may not start the quiz after the deadline has passed.
  - **Maintain your own timer from when you start the quiz.**
  - **Click submit before deadline passes. No Exceptions!**

<table>
<thead>
<tr>
<th>Location</th>
<th>Silicon Valley</th>
<th>Pittsburgh</th>
<th>Rwanda</th>
<th>Adelaide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Apr 23, 09:01 PM</td>
<td>Apr 24, 00:01 AM</td>
<td>Apr 24 06:01 AM</td>
<td>Apr 24 01:31 PM</td>
</tr>
<tr>
<td>Deadline</td>
<td>Apr 24, 08:59 PM</td>
<td>Apr 24, 11:59 PM</td>
<td>Apr 25 05:59 AM</td>
<td>Apr 25 01:29 PM</td>
</tr>
</tbody>
</table>
Quiz 5

• 5% of your Overall Grade
• You only have 1 attempt
• You can save your Quiz answers
  – Highly recommended
  – Save prompt every 15 minutes
• What can I expect from the Quiz?
  – Questions similar to the activities in the Units
  – multiple choice, fill-in-the-blanks, numeric questions,
    ...
• Feedback for Quiz 5 is released after the deadline passes
Upcoming Deadlines

● Quiz 5, Dist. Prog. & Analytics Engines
  ○ **Due:** 11:59PM ET April 24th (Friday)

● P4.2 Iterative Programming using Spark
  ○ **Due:** 11:59PM ET April 26th (Sunday)
Part II: PageRank

• Construct an adjacency list
• Assign initial PageRank values
• Iteratively compute PageRanks

\[ \text{PR}[P_x] = (1 - d) + d \left( \sum_{i=1}^{n} \frac{\text{PR}[P_i]}{C[P_i]} \right) \]
Edge List - Adjacency List

Edge List and Adjacency Lists

- List of edges

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

- Adjacency lists

PageRank in Spark (Scala)
(Note: This is a simpler version of PageRank than P4.2)

```scala
val links = spark.textFile(..).map(s =>
    val parts = s.split("\s+")
    (parts(0), parts(1))
).distinct().groupByKey().persist()
// RDD of (URL, links) as adjacency List

var ranks = links.mapValues(v => 1.0)
// RDD of (URL, rank) as initial PR values

for (i <- 1 to ITERATIONS) {
    // Build an RDD of (targetURL, float) pairs
    // with the contributions sent by each page
    val contribs = links.join(ranks).values.flatMap {
        case (urls, rank) =>
            urls.map(dest => (dest, rank / urls.size))
    }
    // Sum contributions by URL and get new ranks
    ranks = contribs.reduceByKey(_ + _)
        .mapValues(0.15 + 0.85 * _)
}
```
PageRank - Dangling Links 1

Dangling pages are pages without outlinks

In the graph at the right with dangling page, initially

- PR1 = 1.0 PR2 = 1.0 PR3 = 1.0 Sum = 3.0

After computation

- PR1 = 0.15 + 0.85 * (PR2 / 2) = 0.575
- PR2 = 0.15 + 0.85 * (PR1 / 2) = 0.575
- PR3 = 0.15 + 0.85 * (PR1 / 2 + P2 / 2) = 1
- Sum = PR1 + PR2 + PR3 = 2.15

Contribution loss! Summation is NOT constant!
PageRank - Dangling Links 2

Solution: Distribute the weight of dangling pages

Initially

- \( PR_1 = 1.0 \) \( PR_2 = 1.0 \) \( PR_3 = 1.0 \) \( \text{Sum} = 3.0 \)

After computation

- \( PR_1 = 0.15 + 0.85 \times (PR_2 / 2 + PR_3 / 3) = 0.8583 \)
- \( PR_2 = 0.15 + 0.85 \times (PR_1 / 2 + PR_3 / 3) = 0.8583 \)
- \( PR_3 = 0.15 + 0.85 \times (PR_1 / 2 + PR_2 / 2 + PR_3 / 3) = 1.2833 \)
- \( \text{Sum} = PR_1 + PR_2 + PR_3 = 3.0 \)

Summation is constant!