15-319 / 15-619
Cloud Computing

Recitation 13

April 12th 2016

Overview

- **Last week's reflection**
  - Project 4.1
  - Quiz 11
- **Budget issues**
  - Tagging, 15619Project
- **This week's schedule**
  - Unit 5 - Modules 21
  - Project 4.2
  - 15619Project Phase 3
- **Spark 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 4.1 FAQ

- End-to-End Application using MapReduce, H-Base and web frontend
  - Text cleaning -> NGrams -> Language Model
  - Web app querying HBase
  - Extending ideas for word auto-completion
- FAQs
  - Ambiguity in data cleaning
    - Idea: exclude content that is not part of human language
  - Unable to load data into HBase from Reducer, MapReduce program hangs randomly
    - Use the correct jars, learn to manually pack your JAR
    - Test on small datasets first
- Secret to MapReduce: Start small

Module to Read

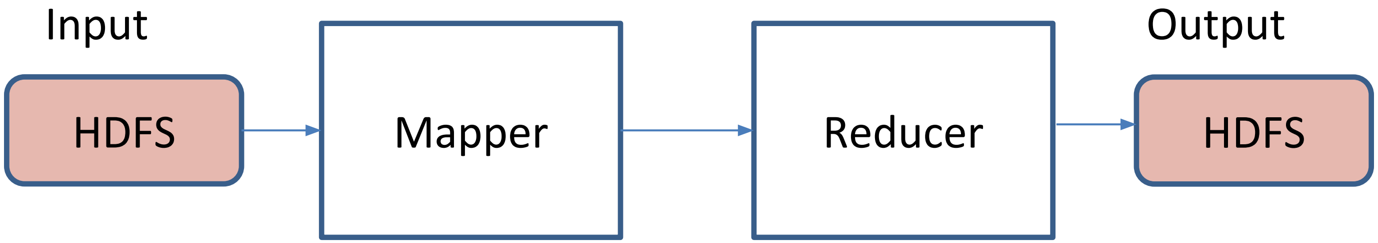
- UNIT 5: Distributed Programming and Analytics Engines for the Cloud
  - Module 18: Introduction to Distributed Programming for the Cloud
  - Module 19: Distributed Analytics Engines for the Cloud: MapReduce
  - Module 20: Distributed Analytics Engines for the Cloud: Spark
  - Module 21: Distributed Analytics Engines for the Cloud: GraphLab (open now)
  - Module 22: Message Queues and Stream Processing (opens on 4/16/2016)

Project 4

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

Typical MapReduce Job

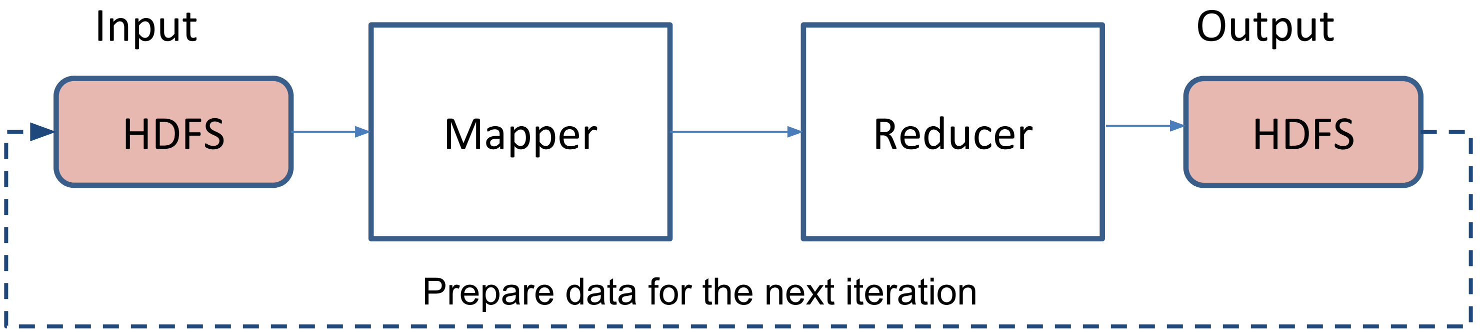
- Simplistic view of a MapReduce job



- 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?

Resilient Distributed Datasets (RDDs)

- RDDs
  - can be in-memory or on disk
  - are read-only objects
  - are 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

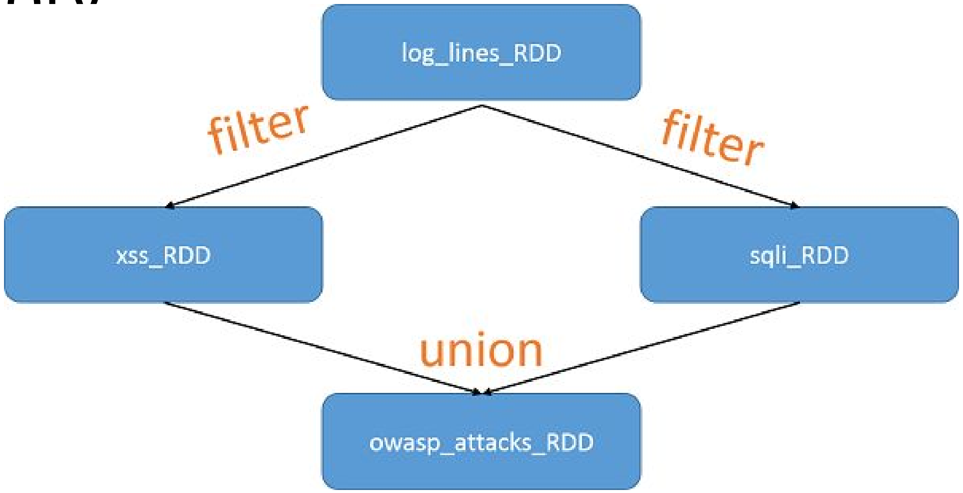
```
>>>transform_RDD =  input_RDD.filter(lambda x: "abcd" in x)
```

- Action
  - Computations on an RDD that return a single object

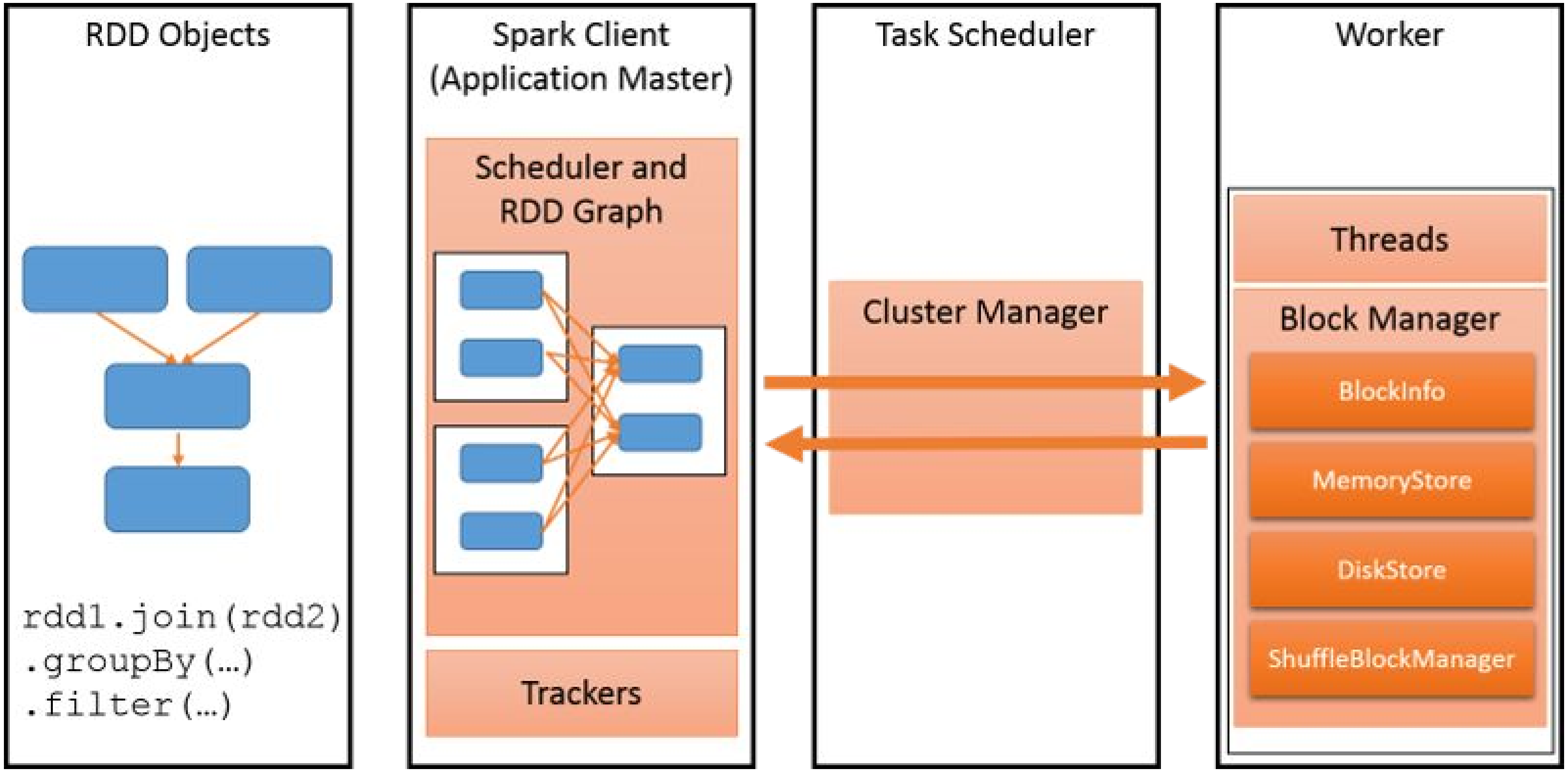
```
>>>print "Number of “abcd”:" + transform_RDD.count()
```

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 can be in-memory



The Spark Framework



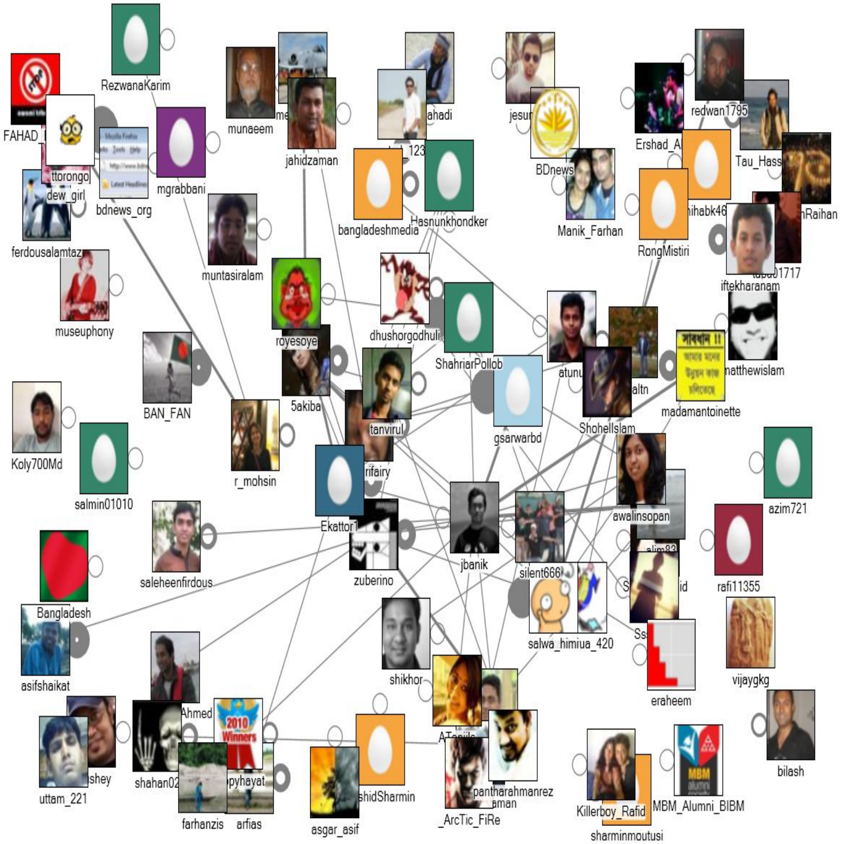
```
rdd1.join(rdd2)
.groupBy(…)
.filter(…)
```

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

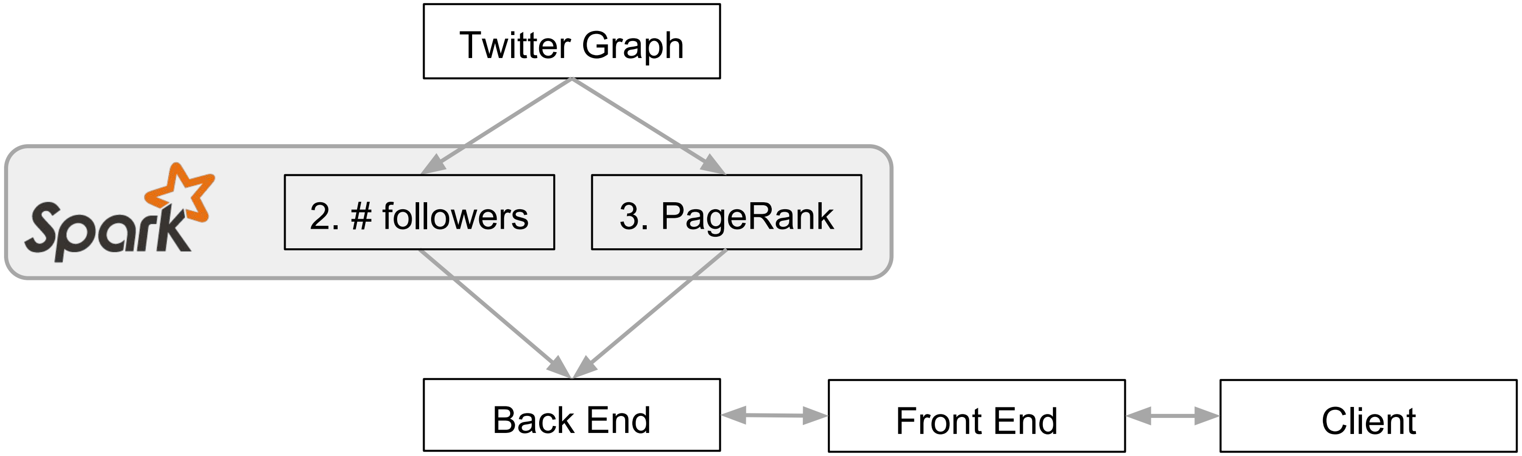
- Use Spark to analyze the Twitter social graph
  - Number of nodes and edges
  - Number of followers for each user
  - Run PageRank to compute the influence of users
  - Fast runs get a bonus



People tweeting with #Shahbag

Project 4.2 - Overview

- Use the Twitter social graph dataset
- Analyze the social graph with Spark
- Find the influence of users and rank them with PageRank



Project 4.2 - Three Parts

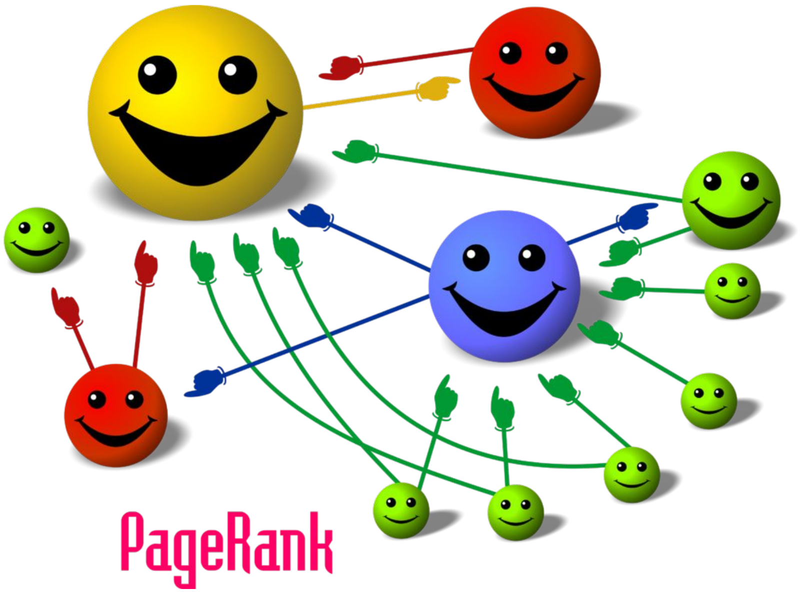
1. Enumerate the Twitter Social Graph
   - Find the number of nodes and edges
   - Edges in the graph are directed. (u, v) and (v, u) should be counted as two edges

2. Find the number of followers for each user

3. Rank each user by influence
   - Run PageRank with 10 iterations
   - Need to deal with dangling nodes

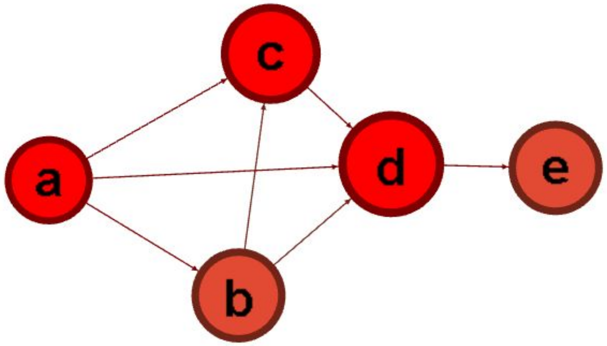
PageRank

- 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-hi-res". Licensed under CC BY-SA 2.5 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:PageRank-hi-res.png#/media/File:PageRank-hi-res.png

PageRank



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

$$\text{Rank}[V_x] = (1-d) + d\left(\sum_{i=1}^{n} \frac{\text{Rank}[V_i]}{C[V_i]}\right)$$

where V_x is Vertex x, d is a damping factor,
and V_i is one of the n neighboring vertices of V_x,
and $C[V_i]$ is the count of the neighbors of Vertex V_i

- Iterate for 10 iterations
- Formula to be implemented for 4.2 is slightly more complex. Read carefully!!!

PageRank in Spark (Scala)

(Note: This is a simpler version of PageRank, than P4.2)

```
val links = spark.textFile(...).map(...).persist()
var ranks = // RDD of (URL, rank) pairs
for (i <- 1 to ITERATIONS)
{
    // Build an RDD of (targetURL, float) pairs
    // with the contributions sent by each page
    val contribs = links.join(ranks).flatMap
    {
        (url, (links, rank)) =>
        links.map(dest => (dest, rank/links.size))
    }

    // Sum contributions by URL and get new ranks
    ranks = contribs.reduceByKey((x,y) => x+y)
                 .mapValues(sum => a/N + (1-a)*sum)
}
```

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
  - 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

- Demo in second half of recitation

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

Grading

- Submit your work in the submitter instance
- Don’t forget to submit your code
- For Task 1
  - Put your answers in the answer file
  - Run submitter to upload your answer
- For Task 2
  - Load your result into the **follower** table in database
  - Run the webserver and use submitter to submit
- For Task 3
  - Load your result into the **pagerank** table in database
  - Run the webserver and use submitter to submit
  - Bonus for execution times < 1800 seconds
    - No shortcuts!!!

GraphLab Bonus

- Additional 5% bonus for running PageRank on GraphLab

- Simply launch a cluster and run the existing implementation of PageRank

- Easiest 5% you’ll ever make

- GraphLab is a part of Quiz 12, so dig in!!!

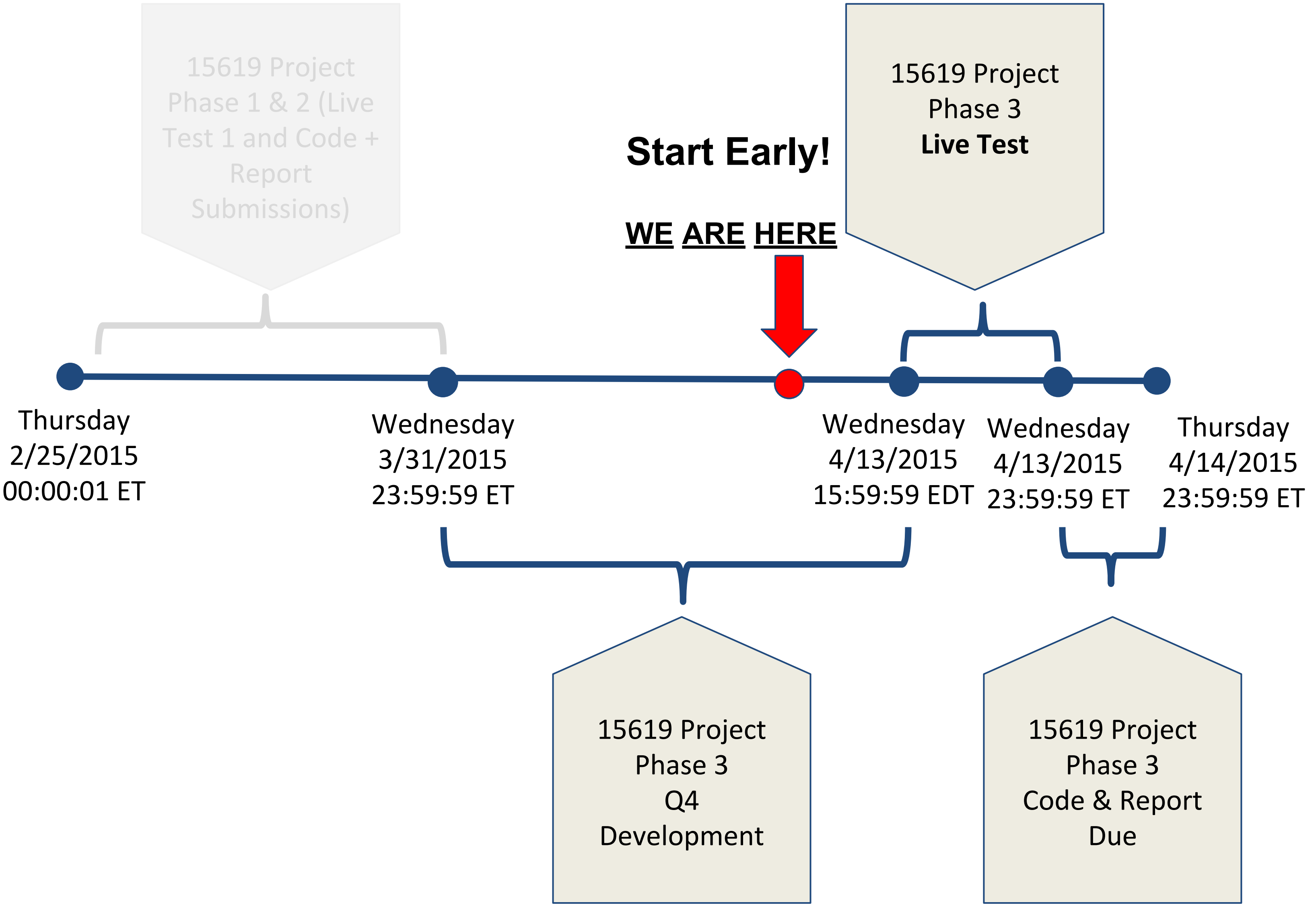
Upcoming Deadlines

- Project 4.2 : Iterative Programming with Spark
  - Due: 04/17/2016 11:59 PM Pittsburgh
- 15619Project : Phase 3
  - Live-test due: 04/13/2016 3:59 PM Pittsburgh
  - Code and report due: 04/14/2016 11:59 PM Pittsburgh

Questions?

TWITTER DATA ANALYTICS: 15619 PROJECT

15619 Project Phase 3 Deadlines



15619 Project Phase 1 & 2 (Live Test 1 and Code + Report Submissions)

Start Early!

WE ARE HERE

15619 Project Phase 3 **Live Test**

Thursday 2/25/2015 00:00:01 ET

Wednesday 3/31/2015 23:59:59 ET

Wednesday 4/13/2015 15:59:59 EDT

Wednesday 4/13/2015 23:59:59 ET

Thursday 4/14/2015 23:59:59 ET

15619 Project Phase 3 Q4 Development

15619 Project Phase 3 Code & Report Due

15619Project Time Table

Phase (and query due)	**Start**	**Deadline**	**Code and Report Due**
Phase 1 Part 1 ● Q1, Q2	Thursday 02/25/2016 00:00:01 EST	Wednesday 03/16/2016 23:59:59 EDT	Thursday 03/17/2016 23:59:59 EDT
Phase 2 ● Q1, Q2, Q3	Thursday 03/17/2016 00:00:01 EDT	Wednesday 03/30/2016 15:59:59 EDT	
Phase 2 Live Test (Hbase/MySQL) ● Q1, Q2, Q3	Wednesday 03/30/2016 18:00:01 EDT	Wednesday 03/30/2016 23:59:59 EDT	Thursday 03/31/2016 23:59:59 EDT
Phase 3 ● Q1, Q2, Q3, Q4	Thursday 03/31/2016 00:00:01 EDT	Wednesday 04/13/2016 **15:59:59 EDT** **(Submit DNS)**	
Phase 3 Live Test ● Q1, Q2, Q3, Q4	Wednesday 04/13/2016 **18:00:01 EDT**	Wednesday 04/13/2016 23:59:59 EDT	Thursday 04/13/2016 23:59:59 EDT

Phase 3

- One last query (Q4)
    - No ETL!
    - Serving write requests
    - Front end caching will not work during the live test
    - Two types of requests, set & get
- **Live Test!**
    - **Warmup, Q1, Q2, Q3, Q4, Mixed Q1-Q4**
        - **Each for 30 min**
    - **Choose HBase or MySQL**
        - **Submit One DNS**

Query 4: Tweet Server

There are five different parameters in the request URL for a request to /q4.

- tweetid (tweet ID)
- op (operation type)
- seq (sequence number)
- fields (comma separated fields involved in the request)
- payload (comma separated payload encoded in Base64)

Execute the requests of a tweetid by the seq (sequence number)

Query 4: Tweet Server

field	type	example
tweetid	long int	15213
userid	long int	156190000001
username	string	CloudComputing
timestamp	string	Mon Feb 15 19:19:57 2016
text	string	Welcome to P4!#CC15619#P3
hashtag	comma separated string	CC15619,P3
ip	string	128.2.217.13
coordinates	string	-75.14310264,40.05701649
repliedby	comma separated userid	156190000001,156190000002,156190000003
reply_count	long int	3
mentioned	comma separated userid	156190000004,156190000005,156190000006
mentioned_count	long int	3
favoritedby	comma separated userid	156190000007,156190000008,156190000009
favorite_count	long int	3
useragent	string	Mozilla/5.0 (iPhone; CPU iPhone OS ...)
filter_level	string	PG-13
lang	string	American

Query 4: Tweet Server

- SET Request `/q4?`
  ```
  tweetid=15213&op=set&seq=1&fields=repliedby,
  reply_count&payload=MzM2NDE5MzE2NjUsMTc0Mjg5OTA1O
  TksOTQ5MDczNzc5NjQsMzkzMjIxMzU4NjQsMTg0NDA4MDg5NT
  UsNTE2MjU1MzMxOTgsOTI4MzA3NTgwNzQ=,Nw==
  ```

- Response
  ```
  TEAMID,TEAM_AWS_ACCOUNT_ID\n
  success\n
  ```

Query 4: Tweet Server

- GET Request `/q4?tweetid=15213&op=get&seq=2&fields=repliedby,reply_count&payload=`

- Response

```
TEAMID,TEAM_AWS_ACCOUNT_ID\n
MzM2NDE5MzE2NjUsMTc0Mjg5OTA1OTksOTQ5MDczNzc5NjQsM
zkzMjIxMzU4NjQsMTg0NDA4MDg5NTUsNTE2MjU1MzMxOTgsOT
I4MzA3NTgwNzQ=\n
Nw==\n
```

Please ensure that you maintain strong consistency for Q4.

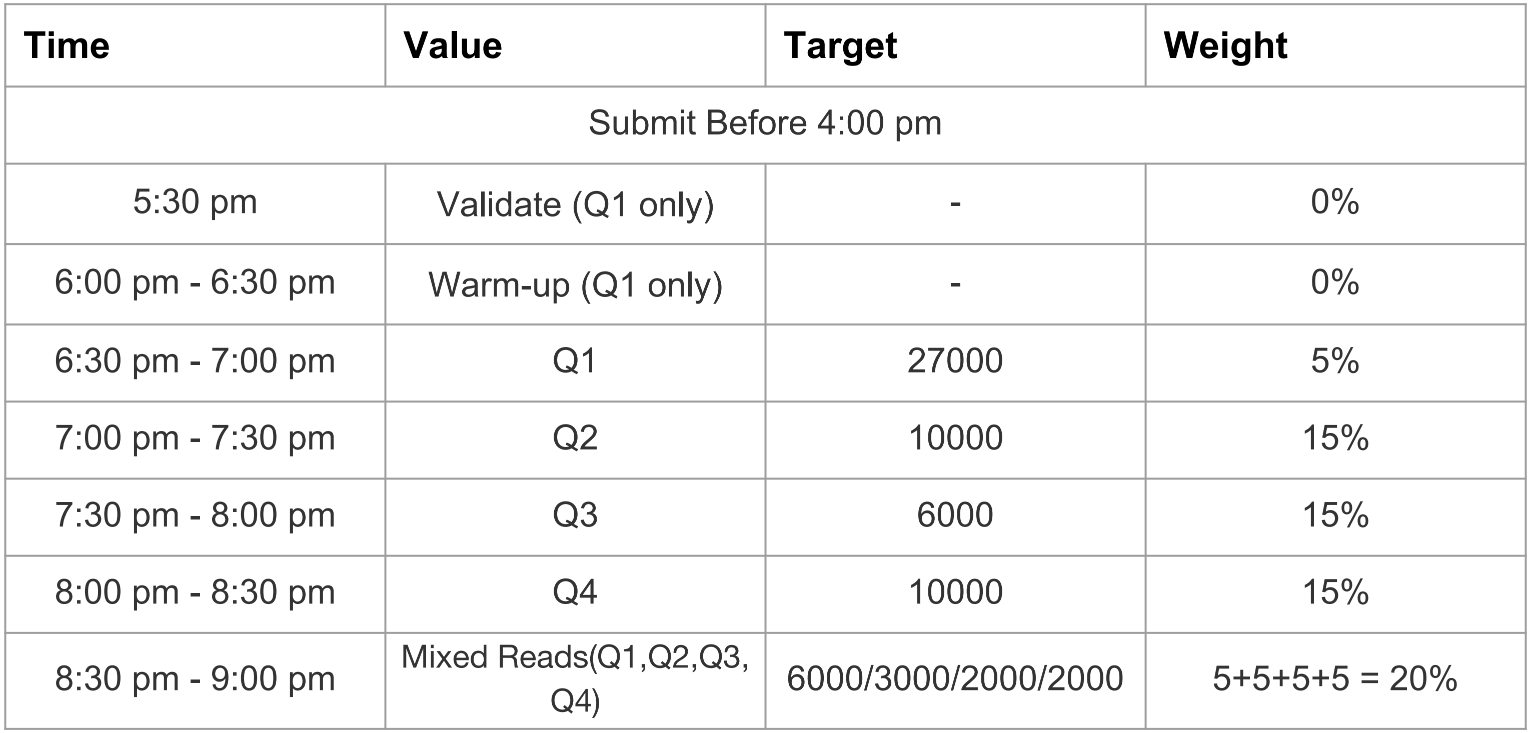
General Hints

- Don’t blindly optimize for every component, identify the bottlenecks using fine-grained profiling
- Use caches wisely: cache in HBase and MySQL is obviously important, storing everything in the frontend cache will lead to failure during the live test
- Review what we have learned in previous project modules
  - Scale out
  - Load balancing
  - Replication and sharding
  - Strong consistency (correctness is very important in Q4)
- Look at the feedback of your Phase 1 and Phase 2 reports!

Q4 Hints

- MySQL DBs behind an ELB may require a forwarding mechanism.
- Consider forwarding the requests but pay attention to latency.
- Consider batch writes.
- Think about effective distributed caching techniques.
- Don’t block your frontend server.
- The `text` field can be as large as 200 characters. Not just 140 characters.

Phase 3 Live Test



Time	**Value**	**Target**	**Weight**
Submit Before 4:00 pm			
5:30 pm	Validate (Q1 only)	-	0%
6:00 pm - 6:30 pm	Warm-up (Q1 only)	-	0%
6:30 pm - 7:00 pm	Q1	27000	5%
7:00 pm - 7:30 pm	Q2	10000	15%
7:30 pm - 8:00 pm	Q3	6000	15%
8:00 pm - 8:30 pm	Q4	10000	15%
8:30 pm - 9:00 pm	Mixed Reads(Q1,Q2,Q3, Q4)	6000/3000/2000/2000	5+5+5+5 = 20%

- Phase 3 report is worth 30% of the Phase 3 grade.
- Change DNS between 4:00pm to 6:00pm will result in 10% penalty
- No changes allowed after 6:00pm

Phase 3 Live Tips

- Warming up your system will be very important
  - EBS warm up, if launching from an AMI
  - ELB warm up
- Pay attention to malformed requests, your system should handle exceptions gracefully
- Not all teams do well in Q1 live test, this is the first thing the you should think of for Phase 3 live test. What is the reason?
- Do NOT use SPOT instances
- Use ELB, or Elastic IP, etc. So that your system will be available when some machines have issues

What’s due soon?

- Phase 3 Development
  - **Submission by 15:59 ET (Pittsburgh) Wed 04/13**
    - **Live Test from 6 PM to 9 PM EDT**
  - Fix Q1 - Q3 if you did not go well
  - New query Q4
  - Phase 3 counts for **60%** of the 15619Project grade
- Phase 3 Report
  - **Submission 23:59:59 ET (Pittsburgh) Thur 04/14**
  - Explain in detail the strategies you used
  - Difficulties you encountered even if you didn’t get a good score