15-319 / 15-619
Cloud Computing

Weekly Overview 7

March 1st, 2022

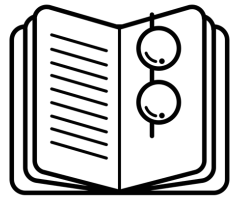
Recap of Last Week's Activities

- Project 2 Discussion
- OLI Unit 3 - Modules 10, 11, 12
- Quiz 5
- Project 3 - Part 1 Released - SQL and NoSQL
  - Extended by two days - Deadline on March 1st 11:59 PM ET
- Team Project, Phase 1 - Microservice 1 Checkpoint

This Week's Activities

- **Project 3, Part 1 Discussion**
  - Due on 8th March 11:59 PM ET
- **OLI Unit 3: Virtualizing Resources for the Cloud**
  - Module 13: Storage and Network Virtualization
- **Quiz 6**
  - Due on 3rd March 11:59 PM ET
- **Project 3 - Part 2: Cloud Storage - Heterogeneous Storage in the Cloud**
  - Due on 6th March 11:59 PM ET
- **Team Project, Phase 1**
  - Microservice 1 Final Due on March 6th 11:59PM ET
  - Microservice 2 Checkpoint Due on March 6th 11:59 PM ET

This Week: Conceptual Content



- **Unit 3: Virtualizing Resources for the Cloud**
  - Module 7: Introduction and Motivation
  - Module 8: Virtualization
  - Module 9: Resource Virtualization - CPU
  - Module 10: Resource Virtualization - Memory
  - Module 11: Resource Virtualization – I/O
  - Module 12: Case Study
  - **Module 13: Storage and Network Virtualization**

OLI Module 13 - Storage Virtualization

- **Unit 3 - Module 13: Storage and network virtualization**
  - Software Defined Data Center (SDDC)
  - Software Defined Networking (SDN)
    - Device virtualization
    - Link virtualization
  - Software Defined Storage (SDS)
    - IOFlow

- **Quiz 6**
  - **Due on March 3rd, which is a Thursday!**

Project 3 Part 2: Heterogeneous Storage on the Cloud

Primers for Project 3, Part 2

- Neo4j Primer
- MongoDB Primer
- MySQL Primer

Neo4j Primer

- **Introduction to Graph Databases**
  - Need for Graph Databases
    - We usually require joins between entities in RDBMSs, and they are expensive to compute. Graph databases store connections alongside data in the model
  - Cypher Query Language (CQL)
    - Create: CREATE, Read: MATCH, Update: SET, Delete: DELETE
- **Table indexing**
  - Single Property vs Composite indexing

MongoDB Primer

- **Compare MongoDB and MySQL**
- **MongoDB Features**
- **MongoDB Technicalities**
  - Documents
  - Collections
- **MongoDB Tutorial to practice:**
  - How to import data into MongoDB
  - Some basic queries with Mongo Shell
  - How to Build index to speed up your query

Project 3, Part 2 Overview

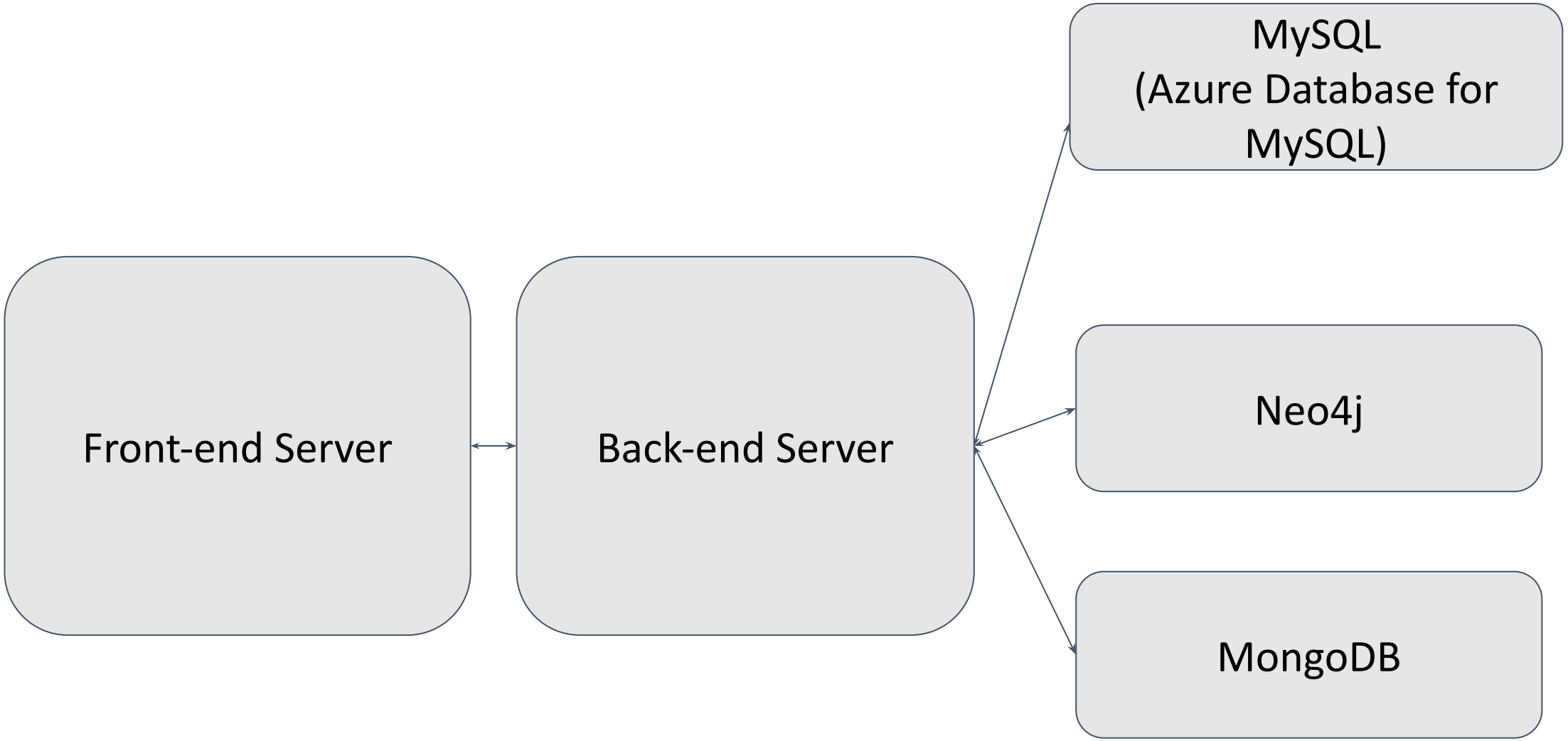
Scenario: Build Your Own Social Network Website using datasets from Reddit.com: **users.csv**, **links.csv**, **posts.json**

- Task 1: Implementing Basic Login with SQL
  - User authentication system : Azure Database for MySQL (users.csv)
  - User info / profile : Azure Database for MySQL
- Task 2: Storing Social Graph using Neo4j
  - Follower, followee : Neo4j (links.csv)
- Task 3: Build Homepage using MongoDB
  - All user generated comments: MongoDB (posts.json)

Project 3, Part 2 Overview (contd.)

- Task 4: Put Everything Together
  - User Timeline: Fanout
- Task 5: Caching
  - Cache the requests with high frequency

Social Network Architecture



Task 1: Implementing Basic Login with MySQL

- Designed to managed highly structured data.
  - Authentication data

- Database-as-a-Service (DBaaS)
  - Azure-managed MySQL database
    - Cloud vendor is responsible for administrative tasks
    - Users are responsible for optimizing applications that use database resources

TDD with Mockito

- Mockito is an open-source testing framework that allows the creation of test double objects (mock objects).
- It is used to mock interfaces so that the specific functionality of an application can be tested without using real resources such as databases, expensive API calls, etc.
- You are required to understand the given implementation, and may use it to quickly debug your solution for Task 1.

Task 2: Storing Social Graph using Neo4j

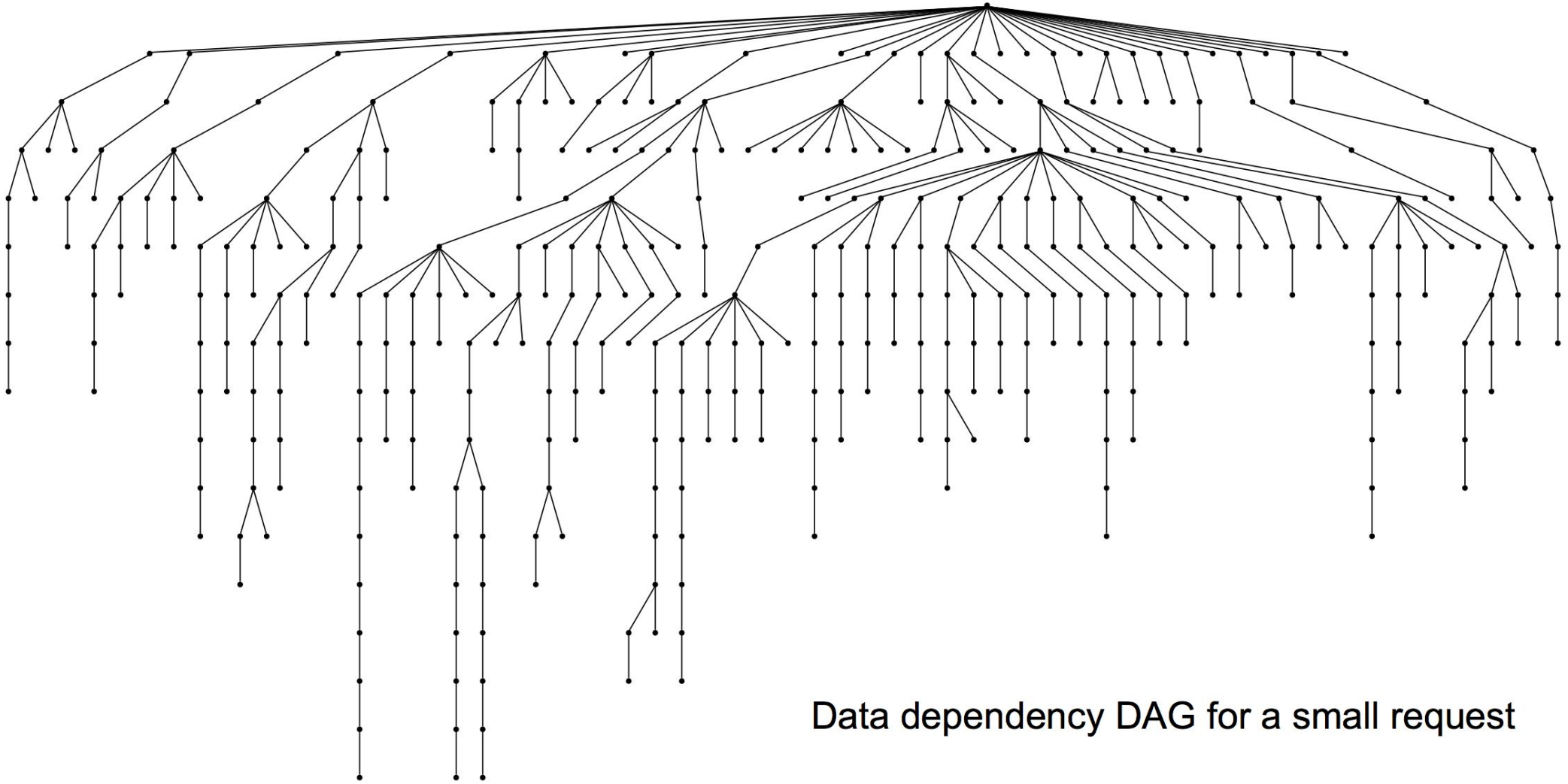
- Designed to treat the relationships between data as equally important as the data
  - Relationships are very important in social graphs
- Property graph model
  - Nodes
  - Relationships
  - Properties
- Cypher query language
  - Declarative, SQL-inspired language for describing patterns in graphs visually

Task 3: Build Homepage using MongoDB

- Document Database
  - Schema-less model
- Highly Scalable
  - Automatically shards data among multiple servers
  - Does load-balancing
- Allows for Complex Queries
  - MapReduce style filter and aggregations
  - Geospatial queries

Task 4: Social Network Timeline

High Fanout in Data Fetching



Data dependency DAG for a small request

Task 4: Social Network Timeline
High Fanout in Data Fetching

- Practice writing complex fan-out queries that span multiple databases:
  - MySQL
  - Neo4j
  - MongoDB

Task 5: Social Network Timeline with Cache

- Fanout and Caching
  - Practice writing complex fan-out queries that span multiple databases.
  - Also practice using a caching mechanism to boost your backend!

P3 - Reminders and Suggestions

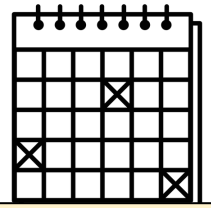
- In Task 4 and 5, you will use the databases from tasks 1 to 3. Make sure to have **all** the databases loaded and ready when working on Task 4 and 5.
- Make sure that you have written **Modular code** for tasks 1 to 3, since you will need to re-use the code in tasks 4 and 5.
- You can submit one task at a time using the submitter. Remember to have your back-end server running when submitting.
- Make sure to terminate **all** resources using “terraform destroy” after the final submission. Double check on the Azure console that all resources were terminated.

Project 3, Part 2 - Reminder!

- By Sunday, March 6th at 11:59 PM
  - Heterogeneous Storage on the Cloud (Part 2) -- Complete the tasks and make a reflection post
- By Tuesday, March 8th at 11:59PM
  - SQL and NoSQL (Part 1) -- Complete discussion tasks
- By Sunday, March 13th at 11:59 PM
  - Heterogeneous Storage on the Cloud (Part 2) -- Finish discussion tasks

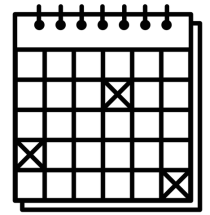
TEAM PROJECT
Twitter Data Analytics

Team Project Time Table



Phase	Deadline (11:59PM ET)
Phase 1 (20%) - **M1** - **M2** - **M3** **(ckpt)**	● M1 CKPT (5%): Sun, 2/27 ● M1 CKPT Report (5%) + Team Intro Form: Sun, 2/27 ● **M1 FINAL (10%): Sun, 3/6** ● **M2 CKPT (5%): Sun, 3/6** ● M2 FINAL (50%): Sun, 3/20 ● M3 CKPT (5%): Sun, 3/20 ● Final Report + Code (20%): Tue, 3/22 **BONUSES:** ● M1 Early Bird Bonus (5%): Sun, 2/27 ● **M2 Early Bird Bonus (5%): Sun, 3/6** ● **M2 Correctness Penalty Waiver: Sun, 3/6** ● M3 Early Bird Bonus (5%): Sun, 3/20 ● M3 Correctness Penalty Waiver: Sun, 3/20

Suggested Tasks for Phase 1



Phase 1 weeks	Tasks	Deadline
Week 1-2 ● 02/14 - 02/27	● Team meeting ● Read Write Up & Report ● Complete M1 code & achieve correctness ● Start writing M2 solution ● Think about M3 database schema	● M1 Checkpoint due on 02/27 ● Checkpoint Report due on 02/27
Week 3 ● **02/28 - 03/06**	● **Optimize for M1 performance** ● **Complete correct M2 code** ● **Start ETL process for M3**	● **M1 final target due on 03/06** ● **M2 Checkpoint due on 03/06**
Week 4-5 ● 03/07 - 03/20	● **Optimize for M2 performance** ● **Finish M3 ETL process** ● Complete M3 code & achieve correctness	● M2 final target due on 03/20 ● M3 Checkpoint due on 03/20 ● Final Report due on 03/22

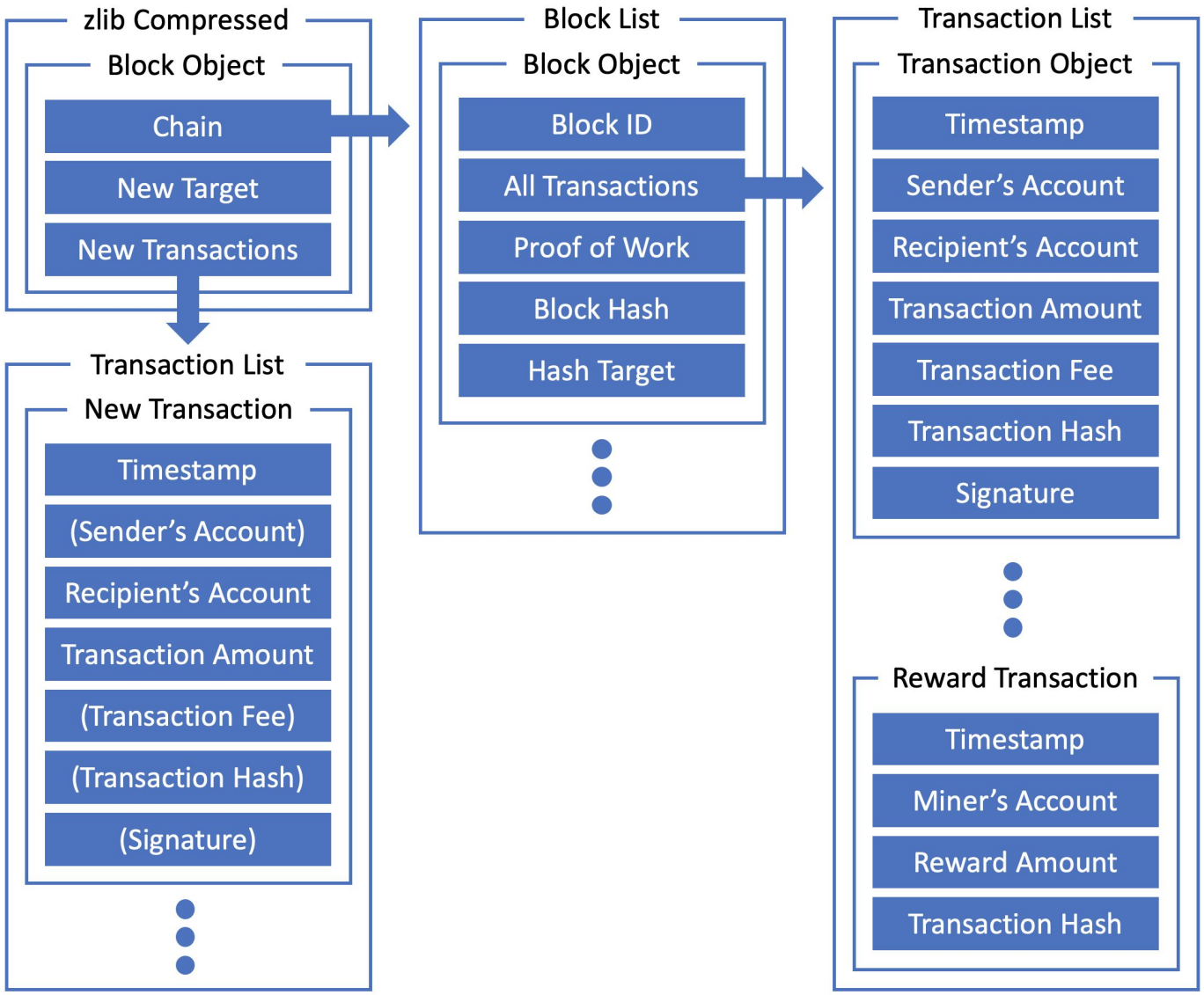
Recap of M1 and M2

- Microservice 1
  - 44/70 teams made a non-zero score 600s submission
  - 16/70 teams achieved 65,000 RPS throughput
- Microservice 2
  - 1/70 teams made a non-zero score 600s submission

M1 Best Teams

Team	QRCode Throughput
MainframeComputing	116045
SnowPear	105291
simple	102791

Microservice 2 Recap




```
{
  "chain": [
    {
      "all_tx": [{
        "recv": 895456882897,
        "amt": 500000000,
        "time": "1582520400000000000",
        "hash": "4b277860"
      }],
      "pow": "0",
      "id": 0,
      "hash": "07c98747",
      "target": "1"
    },
    {
      "all_tx": [
        {
          "sig": 1523500375459,
          "recv": 831361201829,
          "fee": 2408,
          "amt": 126848946,
          "time": "1582520454597521976",
          "send": 895456882897,
          "hash": "c0473abd"
        },
        {
          "recv": 621452032379,
          "amt": 500000000,
          "time": "1582521002184738591",
          "hash": "ab56f1d8"
        }
      ],
      "pow": "202",
      "id": 1,
      "hash": "0055fd15",
      "target": "01"
    },
    {
      "all_tx": [
        {
          "sig": 829022340937,
          "recv": 905790126919,
          "fee": 78125,
          "amt": 4876921,
          "time": "1582521009246242025",
          "send": 831361201829,
          "hash": "46b61f8e"
        },
        {
          "sig": 295281186908,
          "recv": 1097844002039,
          "fee": 0,
          "amt": 83725981,
          "time": "1582521016852310220",
          "send": 895456882897,
          "hash": "b6c1b10f"
        },
        {
          "recv": 905790126919,
          "amt": 250000000,
          "time": "1582521603026667063",
          "hash": "b0750555"
        }
      ],
      "pow": "12",
      "id": 2,
      "hash": "00288a38",
      "target": "0a"
    }
  ],
  "new_target": "007",
  "new_tx": [
    {
      "sig": 160392705122,
      "recv": 658672873303,
      "fee": 3536,
      "amt": 34263741,
      "time": "1582521636327155516",
      "send": 831361201829,
      "hash": "1fb48c71"
    },
    {
      "recv": 895456882897,
      "amt": 34263741,
      "time": "1582521645744862608"
    }
  ]
}
```

Microservice 2 Tips

- My correctness is low
  - Does the sender have enough balance to pay the recipient and the fee?
- How to find PoW?
  - PoW can be any random string as long as it satisfies the hash target
  - for i from 0 to infinity:

    pow = string(i)

    hash = cchash(tx_hash+pow)

    if hash < target: return (pow, hash)

Read M3 Now.
Start ETL Now.

Microservice 3 Checkpoint	-	5%	Sunday, March 20th
Microservice 3 Early Bird Bonus	2000	5%	Sunday, March 20th
Microservice 3 Correctness Bonus	-	Waive one most significant penalty for each team member	Sunday, March 20th
Final Report + Code	-	20%	Tuesday, March 22nd

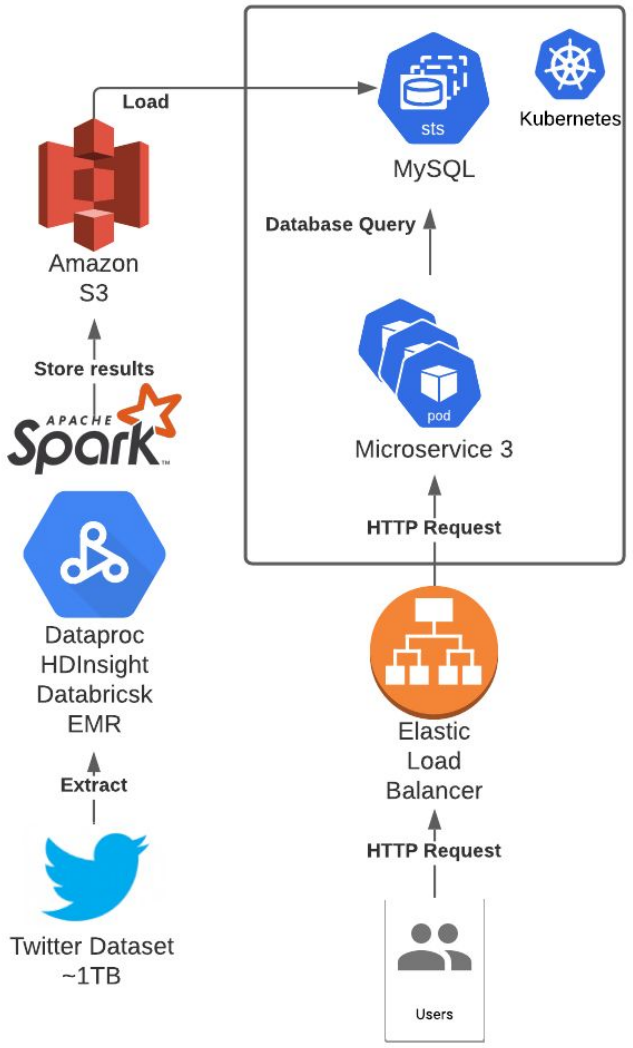
You have two weeks to meet the M3 checkpoint

Question: Is one weekend enough time for M3?

Hint: No. Start now.

Twitter Analytics System Architecture

- Building a performant web service
- Dealing with large scale real world tweet data
- HBase and MySQL optimization



M3 - User Recommendation System

Target throughput: 10,000 RPS **for both MySQL and HBase**

Use Case: Recommend User B, C and D when you follow User A on twitter

Three Scores when making recommendation:

- Interaction Score - closeness
- Hashtag Score - common interests
- Keywords Score - match specific interests

Final Score: Interaction Score * Hashtag Score * Keywords Score

Query: GET /twitter?**user_id**=<ID>&**type**=<TYPE>&**phrase**=<PHRASE>&**hashtag**=<HASHTAG>

Response:

<TEAMNAME>,<AWSID>\n
uid\tname\tdescription\ttweet\n
uid\tname\tdescription\ttweet

M3 - Filtering

Each line from the provided files is a tweet object

- Malformed JSON Object
- Malformed Tweets
- Each tweet must contain valid
  - Tweet ID
  - Sender’s user ID
  - Timestamp
  - Content
  - At least 1 hashtag
- Tweets not in the required languages
- Duplicate Tweets

M3 - Contact Tweets

Given a valid tweet JSON object `t`.

A **contact tweet** is a tweet that is either a reply tweet or a retweet.

- A tweet is a **reply** tweet if `t.in_reply_to_user_id` is not null.
- A tweet is a **retweet** if `t.retweeted_status` is not null.

tweet_id	user_id	content	reply_to_id	retweet_to_id
01	15618	cloud	15213	null
02	15640	computing	null	15319
03	15513	is	15213	null
04	15513	fun	null	null

Then we have the followings:

user_id	contact_tweet_id	contacted_user
15213	01, 03	15618, 15513
15513	03	15213
15319	02	15640
15618	01	15213
15640	02	15319

M3 - User Information

Given a valid tweet JSON object `t`.

User information can appear in `t` and `t.retweeted_status` objects.

- For any tweet `t`, we can find the sender information in `t.user`
- If the tweet `t` happens to be a **retweet**, we can additionally find the original poster's information in `t.retweeted_status.user`

For each user appeared, we can get the timestamp from `t.created_at`.

After processing all the valid tweets, we can get the latest information of all users.

Note: For user information with the same timestamp, break the tie by tweet ID in **descending numerical order**.

M3 - Interaction Score

- Two types of interaction: Retweet and Reply

- **Interaction score** =
  `log(1 + 2 * reply_count + retweet_count)`

Examples:

1. A replied B 4 times; B retweeted A 3 times
   `log(1 + 2*4 + 1*3) = 2.485`
2. A replied B twice; B replied A once
   `log(1 + 2*(2+1) + 1*0) = 1.946`
3. A retweeted B once
   `log(1 + 2*0 + 1*1) = 0.693`
4. no replies/retweets between A and B
   `log(1 + 2*0 + 1*0) = 0`

M3 - Hashtag Score

- `same_tag_count` = hashtags among all the tweets two users **posted**, excluding popular hashtags from the list provided by us.

The final `hashtag_score` is calculated as follows.

- If `same_tag_count > 10`, `hashtag_score = 1 + log(1 + same_tag_count - 10)`.
- Else, `hashtag_score = 1`

For the cases of self-reply or self-retweet , the **hashtag score** will **always be 1**.

Note: hashtags are case-insensitive

Here are a few examples. Assume hashtag `zipcode` is a very popular hashtag that we exclude.

sender_uid	hashtags
15619	Aws, azure, ZIPCODE
15619	Cloud, Azure
15619	Cloud, GCP
15619	cloud, aws
15319	cmu, us
15319	AZure
15319	Cloud, GCP
15319	aWs, zipcode, CLOUD
15513	cmu, us
15513	haha, ZIPcode
15513	zipcode

Given all the tweets above, the hashtag score of the user pairs below are:

uid_1	uid_2	same_tag_count	explanation
15619	15319	13	aws=3, cloud=5, azure=3, GCP=2
15619	15513	0	no match
15319	15513	4	cmu=2, us=2

M3 - Keyword Score

Counting the total number of matches of phrase and also hashtag (both provided in the query) across the **contact tweets** of a specific *type*. The *type* is given in the query, and valid values are `[reply|retweet|both]`.

Matching rule for the phrase: case sensitive match. Example: `haha`

- **`haha`**`ha` has 2 matches (beware: overlapping matches are possible)
- `haHaha` has no matches
- `Haha bahaha` has 1 match

Matching rule for the hashtag: case insensitive exact match. Example: `cloud`

Tweet having hashtags **`#Cloud #CLOUD #CLOUD`** `#cmu`
Note that duplicate tags are allowed, thus `number_of_matches += 3`.

If there are no contact tweets of the type specified in the query,
`keywords_score = 0`.

Otherwise, `keywords_score = 1 + log(number_of_matches + 1)`.

M3 - Final Score and Ordering

Final Score

```
final_score = interaction_score * hashtag_score * keywords_score
```

- Keep 5 decimal points of precision rounding half up **before** ranking
- Ignore user pairs with a final score of 0

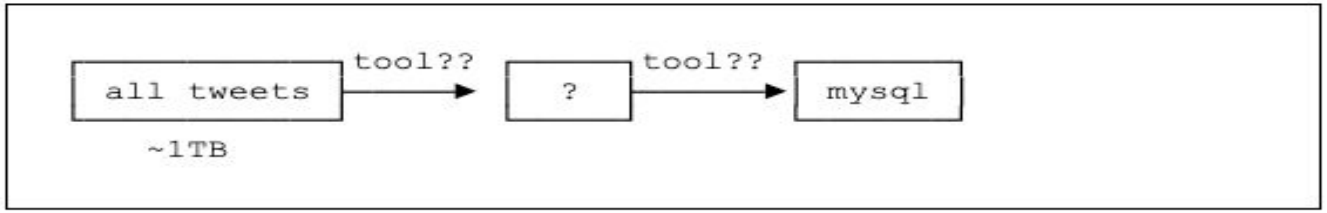
Ordering

- **Rank by the score in descending order**.
  - Break ties by user ID in **descending numerical order**.

For the latest contact tweets between two users, break the tie by tweet ID in **descending numerical order** if they have the same timestamp.

M3 Roadmap

- Use a flowchart as an ETL mind map and code design



- Do the filtering on the first part (part-00000) of the dataset and make sure the result is **exactly the same** as the reference answer
- Start ETL on the mini dataset locally or in GCP/Azure with sufficient unit tests
    - Zeppelin can be your friend
    - Think about what information is necessary for one query
    - Start with a tentative schema and adjust it accordingly
    - Pick some test queries that can help you partially verify the ETL process
    - Store your ETL result as TSV/CSV files
    - Make use of the mini-ref server

Continued ….

M3 Roadmap (continued)

- Start ETL on the entire dataset in GCP/Azure and compare your result against the reference server.
  - Store your ETL results as TSV/CSV and use tools to import into database
  - E - T - L - Verify using the provided reference server
  - Spot a bug, correct the code and rerun the ETL
    - Can you just rerun the ETL partially? Store intermediate results?
  - Make multiple 600s submissions if you think the result is good enough
- Optimize your implementation to reach the target throughput.
  - Identify the potential bottlenecks with profiling
    - Web framework? Query Processing Logic? DB Schema?
  - If you decide to change the schema (you will very likely **need** to do this)
    - Which part of ETL do you need to rerun?
    - If the computation takes X hours and loading all the data into database takes Y hours, how many iterations can you do?
    - Can you accelerate your design iteration?

Spark, Scala and Zeppelin Primers

- Primers for Apache Spark/Scala/Zeppelin are now available
- Learn more about Spark in OPE, Project 4 and OLI Module 20
- Spark stores data in **memory**, allowing it to run an order of magnitude **faster** than Hadoop
- Spark is **more expressive** for some operations
- You can use Spark or Hadoop - it is your choice since you have total freedom in ETL frameworks

Reminders on penalties

- Self-managed Kubernetes cluster + optional EMR, consisting of M family instances **only**, smaller than or equal to **large** type
- Other types are allowed (e.g., t2.micro) **but only for testing**
  - Using these for final submissions = 100% penalty
- Only General Purpose (gp2) SSDs are allowed for storage
  - e.g **m5d is not allowed** since it uses NVMe storage
- AWS endpoints only (EC2/ELB).
- **$0.70/hour (MySQL) and $1.10/hour (HBase)** applies to every submission

Best Wishes!!!