15-319 / 15-619
Cloud Computing

Recitation 7

P3.1 & Team Project

February 25, 2020

Overview

- **Last week's reflection**
  - OLI Unit 3 - Modules 10, 11, 12
  - Quiz 5
  - Project 2.3

- **This week's schedule**
  - OLI Unit 3 - Module 13
  - Quiz 6
  - Project 3.1
  - Team Project, Phase 1 Q1 Checkpoint

Last Week

- **Unit 3: Virtualizing Resources for the Cloud**
  - **Module 10:** Resource Virtualization - Memory
  - **Module 11:** Resource Virtualization – I/O
  - **Module 12:** Case Study
- **Quiz 5**
- **Project 2.3: Functions as a Service (FaaS)**
  - **Task 1:** Explore functions on various CSPs
    - Azure Functions, GCP Cloud Functions, AWS Lambda
  - **Task 2:** Extract thumbnails from a video stream
    - Azure Functions and FFmpeg
  - **Task 3:** Get image labels and index
    - Azure Computer Vision, Azure Search

This Week

- **Unit 3: Virtualizing Resources for the Cloud**
  - **Module 13:** Storage and network virtualization
- **Quiz 6**
- **Project 3: Storage and DBs on the cloud**
  - **Project 3.1:** Files v/s Databases
    - Flat files
    - MySQL
    - Redis & Memcached
    - HBase
- **Team Project**
  - **Phase 1** released.
  - **Q1 Checkpoint** due at the end of this week.

This Week's 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**

Open Learning Initiative
Transforming higher education through the science of learning.

This Week’s Conceptual Content

- **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**

Open Learning Initiative
Transforming higher education through the science of learning.

This Week's Individual Project

- **Project 3: Storage and DBs on the cloud**
  - **P3.1: Files and Databases**
    - Comparison and usage of Flat files, RDBMS (MySQL) and NoSQL (Redis, HBase)
  - **P3.2: Social Networking Timeline with Heterogeneous Backends**
    - Heterogeneous Backends (MySQL, Neo4j, MongoDB, S3)
  - **P3.3: Replication and Consistency**
    - Multi-threaded Programming and Consistency

Project 3

MySQL™

mongoDB

APACHE HBASE

neo4j

SQL

Primers for Project 3

- **Project 3: Storage and DBs on the cloud**
  - **P3.1: Files and Databases**
    - Primer: MySQL
    - Primer: Storage & IO Benchmarking
    - Primer: NoSQL
    - Primer: HBase basics
  - **P3.2: Social Networking Timeline with Heterogeneous Backends**
    - Primer: MongoDB
  - **P3.3: Replication and Consistency**
    - Primer: Introduction to Consistency Models
    - Primer: Introduction to multithreaded programming in Java

MySQL Primer

- **Introduction to *Structured Query Language* (SQL)**
  - SELECT
  - JOIN
  - GROUP BY
  - CREATE, ALTER, DROP, INSERT, UPDATE, DELETE
- **Table indexing**
  - Single column vs Multi-column indexing
  - Common pitfalls
- **Storage Engines**
  - MyISAM
  - InnoDB

Storage Engines in MySQL

- A storage engine is a software module that a DMS uses to create, read, update data from a database

- **MyISAM** and **InnoDB**
- They have:
  - Different caching mechanisms
  - Different locking mechanisms
  - Are optimized for either read or write
  - More differences are explained in the primer

Experiment, and think of which one to use in the team project

Read the MySQL primer

Storage & IO Benchmarking

- **Run sysbench**
  - Use *prepare* to load data for testing
- **Experiments**
  - Run sysbench with different storage systems and instance types
  - Do this multiple times to reveal different behaviors and results
- **Compare Requests Per Second (RPS)**

Remember to tag resources with the current project's tags

Performance Benchmark Sample Report

Scenario	Instance Type	Storage Type	RPS Range	RPS Increase Across 3 Iterations
1	t3.micro	EBS Magnetic Storage	171.12, 172.33, 189.34	Trivial (< 5%)
2	t3.micro	EBS General Purpose SSD	1649.65, 1709.24, 1729.24	Trivial (< 5%)
3	m4.large	EBS Magnetic Storage	527.70, 973.63, 1246.67	Significant (can reach ~140% increase with an absolute value of 450-700)
4	m4.large	EBS General Purpose SSD	2046.66, 2612.00, 2649.66	Noticeable (can reach ~30% increase with an absolute value of 500-600)

IO Benchmarking Conclusions

- **SSD has better performance than magnetic disk**
- **m4.large instance offers better performance than t3.micro**
- **The RPS increase across 3 iterations for m4.large is more significant than that for t2.micro**
  - An instance with more memory can cache more of the previous requests for repeated tests
  - Caching is a vital performance tuning mechanism

Project 3.1 Overview

- **Task 1: analyze data in flat files**
  - Linux tools (e.g. grep, awk)
  - Data libraries (e.g. pandas)
- **Task 2: Explore a SQL database (MySQL)**
  - Load data, run queries, indexing, auditing
  - Plain-SQL vs ORM
- **Task 3: Implement a Key-Value Store**
  - Prototype of Redis using TDD
- **Task 4: Explore a NoSQL DB (HBase)**
  - Load data, design key, run basic queries

Refer to the HBase Basics and NoSQL Primers!

Flat Files

- **Flat files, plain text or binary**
  - Comma-Separated Values (CSV)

    ```
    Carnegie,Cloud Computing,A,2018
    ```

  - Tab-Separated Values (TSV)

    ```
    Carnegie\tCloud Computing\tA\t2018
    ```

  - A custom and verbose format

    ```
    University: Carnegie, Course: Cloud
    Computing, Section: A, Year: 2018
    ```

Flat Files

- Lightweight, Flexible, in favor of small tasks
  - Run it once and throw it away
- Inconvenient to perform complicated analysis
- Usually flat files should be fixed or append-only
- Writing to files without breaking data integrity is difficult
- Managing the relations among multiple files is also challenging

Databases

- A collection of organized data
- Database management system (DBMS)
  - Interface between user and data
  - Store/manage/analyze data
- Relational databases
  - Based on the relational model (schema)
  - MySQL, PostgreSQL
- NoSQL Databases
  - Unstructured/semi-structured
  - Redis, HBase, MongoDB, Neo4J

Databases

- **Advantages**
  - Logical and physical data independence
  - Concurrency control and transaction support
  - Query the data easily (e.g., SQL)
- **Disadvantages**
  - Cost (computational resources, fixed schema)
  - Maintenance and management
  - Complex and time-consuming to design schema

Flat Files vs Databases

- **Compare flat files to databases**
- **Think about:**
  - What are the advantages and disadvantages of using flat files or databases?
  - In what situations would you use a flat file or a database?
  - How to design your own database? How to load, index and query data in a database?

Dataset

- **Analyze Yelp's Academic Dataset**
- https://www.yelp.com/dataset_challenge
  - business
  - checkin
  - review
  - tip
  - user

Inspect and visualize data using Facets

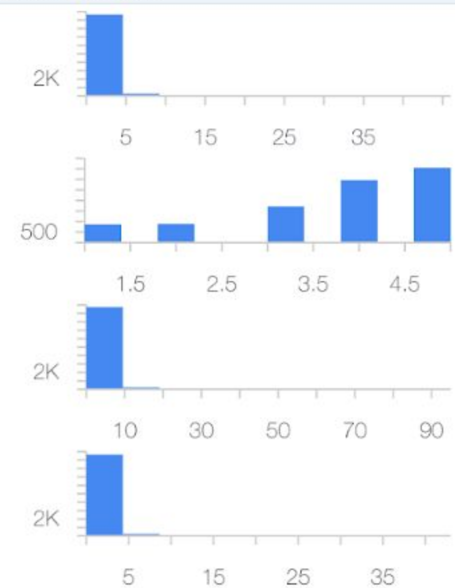
Sort by
Feature order
☐ Reverse order
Feature search

Features: ☑ int(4) ☑ string(5)

Numeric Features (4)

	count	missing	mean	std dev	zeros	min	median	max
funny	10,000	0%	0.65	1.84	**73.27%**	0	0	46
stars	10,000	0%	3.74	1.27	0%	1	4	5
useful	10,000	0%	1.67	3.06	**44.16%**	0	1	95
cool	10,000	0%	0.78	1.96	**65.98%**	0	0	43

Chart to show
Standard
☐ log ☐ expand



Task 1: Flat Files

- **Answer questions in runner.sh**
  - Use tools such as awk and pandas
  - Similar to what you did in Project 1
- **Merge TSV files by joining on a common field**
- **Identify the disadvantages of flat files**

You may use Jupyter Notebook to help you solve the questions in Python

Task 2: MySQL

- Prepare tables
  - A script to create the table and load data is provided
- Write MySQL queries to answer questions
- Learn JDBC
- Complete MySQLTasks.java
- Aggregate functions, joins
- *Statement* and *PreparedStatement*
- SQL injection
- Learn how to use proper indexes to improve performance

MySQL Indexing

- **Schema**
  - The structure of the tables and the relations between tables
  - Based on the structure of the data and the application requirements

- **Index**
  - An index is simply a pointer to data in a table
  - It is a data structure (lookup table) that helps speed up the retrieval of data from tables (e.g., B-Tree, Hash indexes, etc.)
  - Based on the data as well as queries
  - Build indexes based on the types of queries you'll expect

We have an insightful section about the practice of indexing, read it carefully! **Very helpful for the team project**

EXPLAIN statement in MySQL

- **How do we evaluate the performance of a query?**
  - Run it

- **What if we want/need to predict the performance without execution?**
  - Use EXPLAIN statement

- **The EXPLAIN statement on a query predicts:**
  - The number of rows to scan
  - Whether it makes use of indexes or not

Object Relational Mapping (ORM)

- **ORM abstracts the interaction with a DB for you:**
  - Maps the domain class with the database table
  - Map each field of the domain class with a column of the table
  - Map instances of the classes (objects) with rows in the corresponding tables

	Mapped to	
public class Course {	→	course
String courseId;	→	course_id (PK)
String name;	→	name
}		
Domain Class	→	Database Table
Objects	→	Rows

Benefits of ORM

- **Decoupling of responsibilities**
  - ORM decouples the CRUD operations and the business logic code
- **Productivity**
  - No need to keep switching between your OOP language such as Java/Python, etc. and SQL
- **Flexibility to meet evolving business requirements**
  - Cannot eliminate the schema update problem, but it may ease the difficulty, especially when used together with data migration tools
- **Persistence transparency**
  - Changes to a persistent object will be automatically propagated to the database without explicit SQL queries
- **Vendor independence**
  - Abstracts the application from the underlying SQL database and SQL dialect

ORM Question in the MySQL Task

- The current business application exposes an API that returns the most popular Pittsburgh businesses
- It is based on a SQLite3 database with an outdated schema
- **Your task:**
  - Plug the business application to the MySQL database and update the definition of the domain class to match the new schema
- The API will be backwards compatible without modifying any business logic code

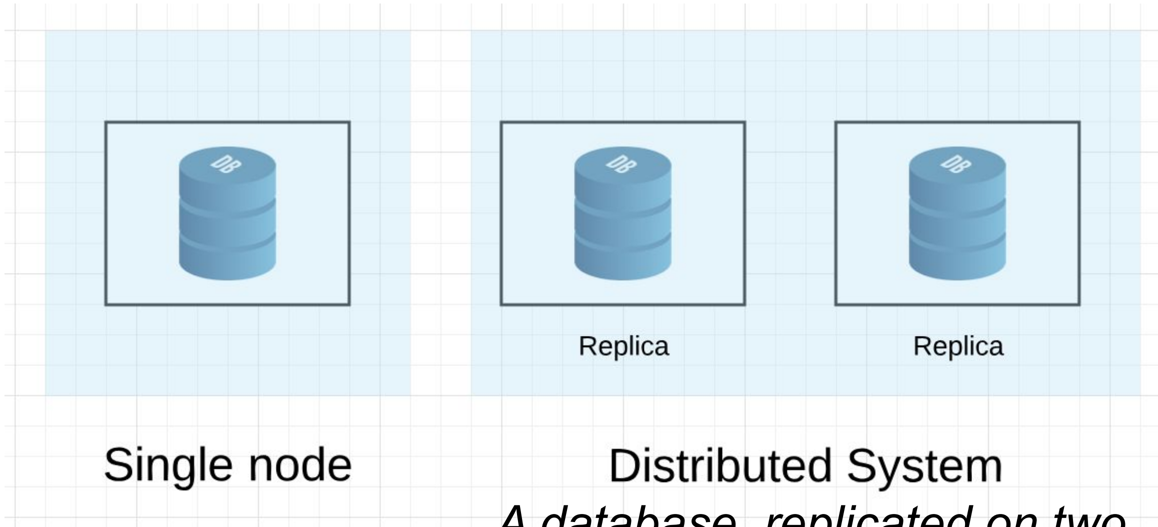
NoSQL

- **Non-SQL or NotOnly-SQL**
  - Non-relational
- **Why NoSQL if we already have SQL solutions?**
  - Flexible data model (schemaless, can change)
  - Designed to be distributed (scale horizontally)
  - Certain applications require improved performance at the cost of reduced data consistency (data staleness)
- **Basic Types of NoSQL Databases**
  - Schema-less Key-Value Stores (Redis)
  - Wide Column Stores (Column Family Stores) (HBase)
  - Document Stores (MongoDB)
  - Graph DBMS (Neo4j)

CAP Theorem

- It is impossible for a distributed data store to provide all the following three guarantees at the same time:
  - **Consistency:** no stale data
  - **Availability:** no downtime
  - **Partition Tolerance:** network failure tolerance in a distributed system

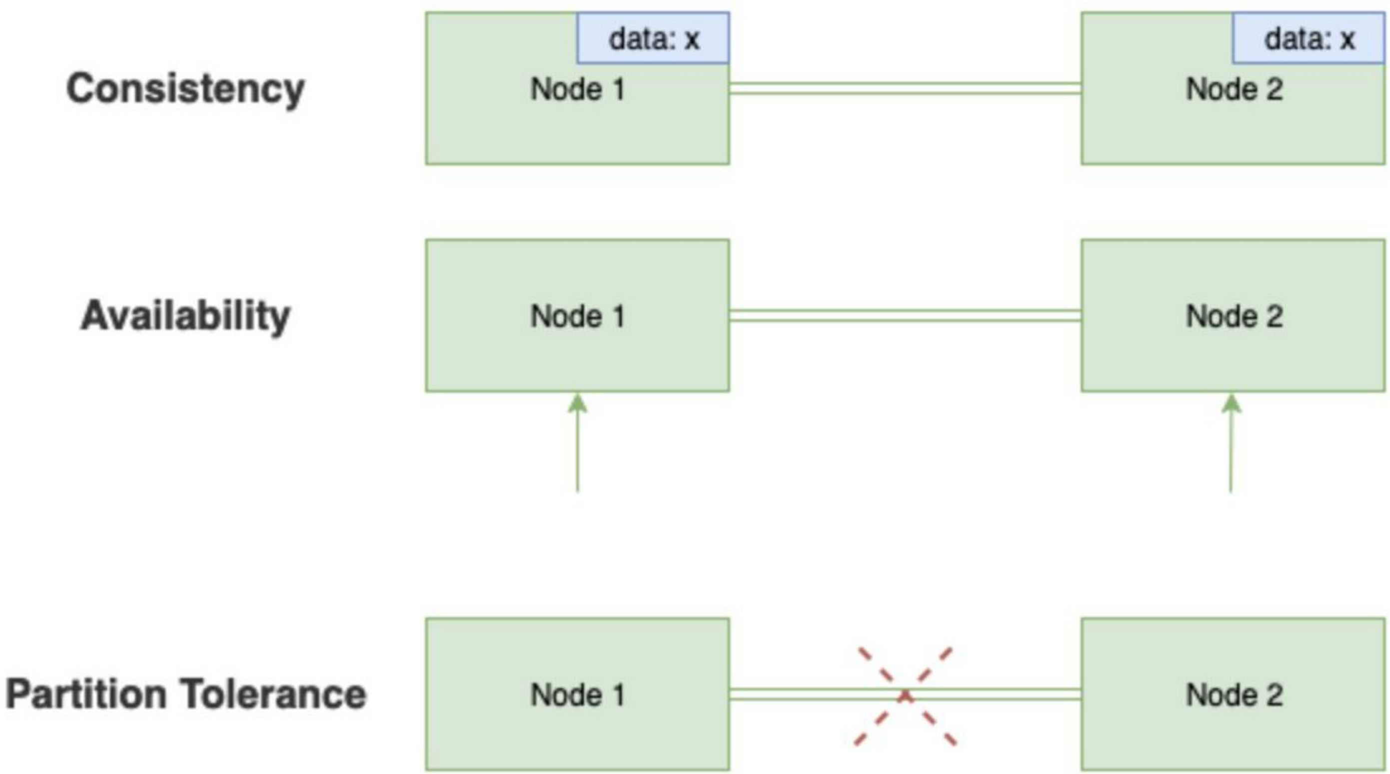
Single Node to Distributed Databases



A database, replicated on two nodes, Node 1 and Node 2

- Since DB is replicated, how is consistency maintained?
- Since the data is replicated, if one replica goes down, will the entire service go down?
- How will the service behave during a network failure?

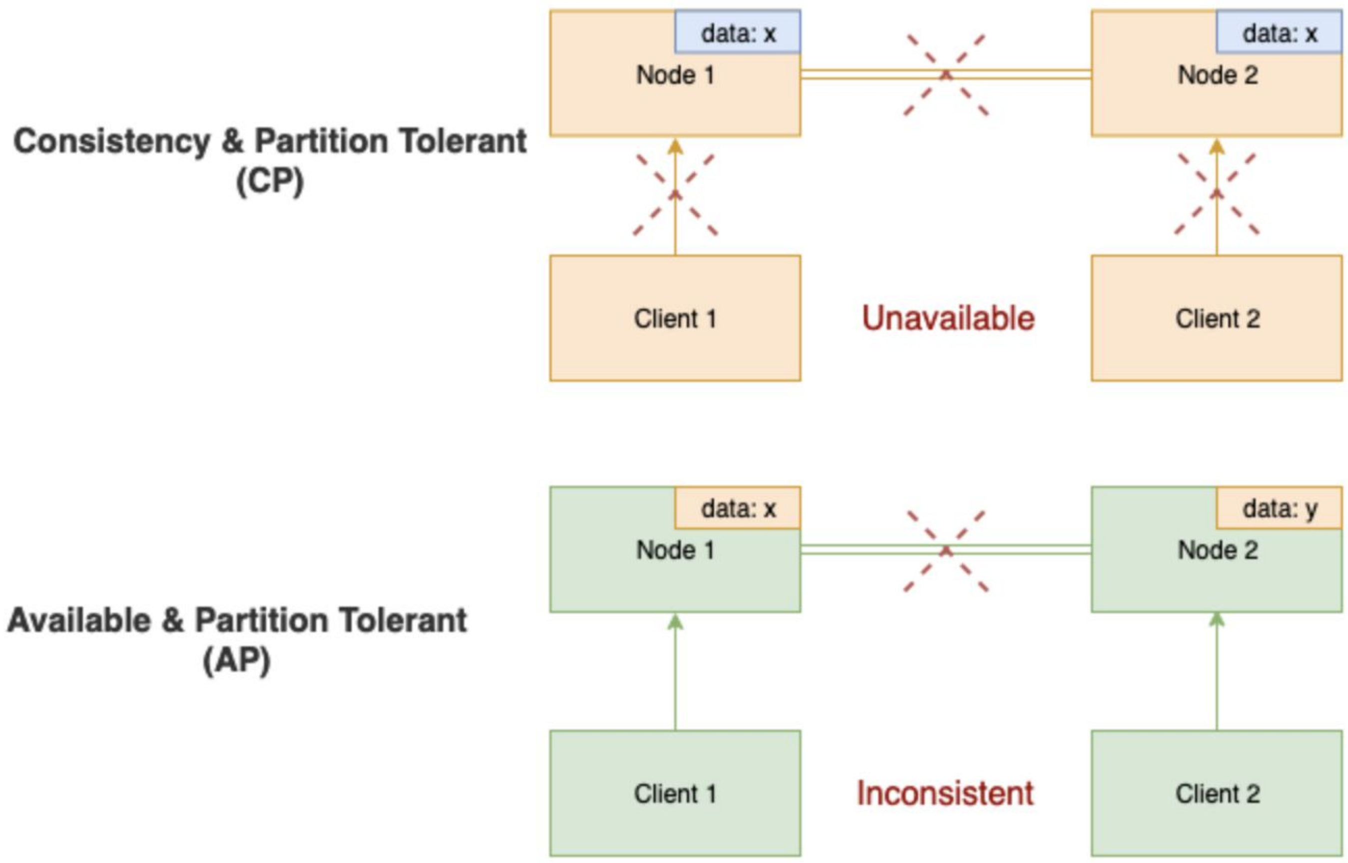
CAP Theorem in Distributed Databases



CAP Theorem

- Only two out of the three are feasible:
  - **CA: non-distributed (MySQL, PostgreSQL)**
    - Traditional databases like MySQL and PostgresQL have only one server
    - Don’t provide partition tolerance
  - **CP: downtime (HBase, MongoDB)**
    - Stop responding if there is partition
    - There will be downtime
  - **AP: stale data (Amazon DynamoDB)**
    - Always available
    - Data may be inconsistent among nodes if there is a partition

Only two at a time



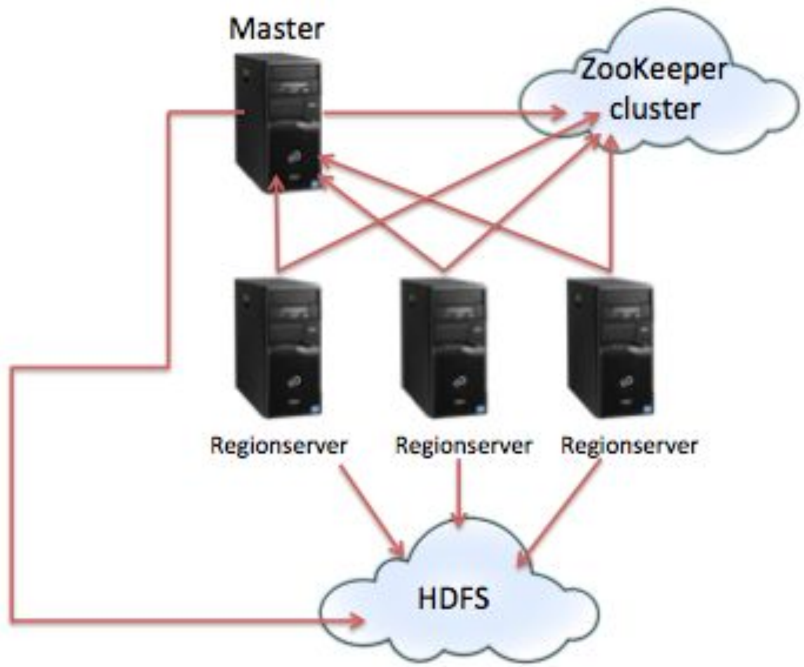
Task 3: Implement Redis

- Key-value store is a type of NoSQL database
    - Redis
    - Memcached
- Widely used as an in-memory cache

- **Your task:**
    - Implement a simplified version of Redis
    - We provide starter code *Redis.java*
    - You will implement
        - Hashes and Lists data structures in Redis
    - **TDD with 100% code coverage**

Task 4: Explore HBase

- HBase is an open source, column-oriented, distributed database developed as part of the Apache Hadoop project



- **Refer to the HBase Basics Primer**

RowKey Design

- Rows in HBase are sorted lexicographically by row key
- **Hotspotting**
  - A large amount of client traffic is directed to one/few node/s
    - Pre-split the table
    - A good key design is very important
      - Salting: randomly assign prefix

        foo0001 → a-foo0001

        foo0002 → d-foo0002

        foo0003 → b-foo0003

        foo0004 → b-foo0004

      - Hashing: deterministically assign prefix

        hash(foo0001) % NUM_REGIONS== 5 → 5-foo0001

Task 4: Explore HBase

- **Your task:**
  - Launch an HDInsights cluster
  - Load data so that it is evenly distributed across regions
    - **Make sure to submit a *design.pdf* file with your key design**
  - Try different commands in the *hbase-shell*
  - Complete *HBaseTasks.java* using HBase Java APIs

Project 3.1 - Reminders

- **Tag your resources:**
  - **Key: Project, Value: 3.1**
- An HDInsight cluster is very expensive
  - Exercise caution to plan for the budget
- Provisioning an HDInsight cluster takes ~30min
- Loading data to MySQL takes ~40 minutes
  - Be patient

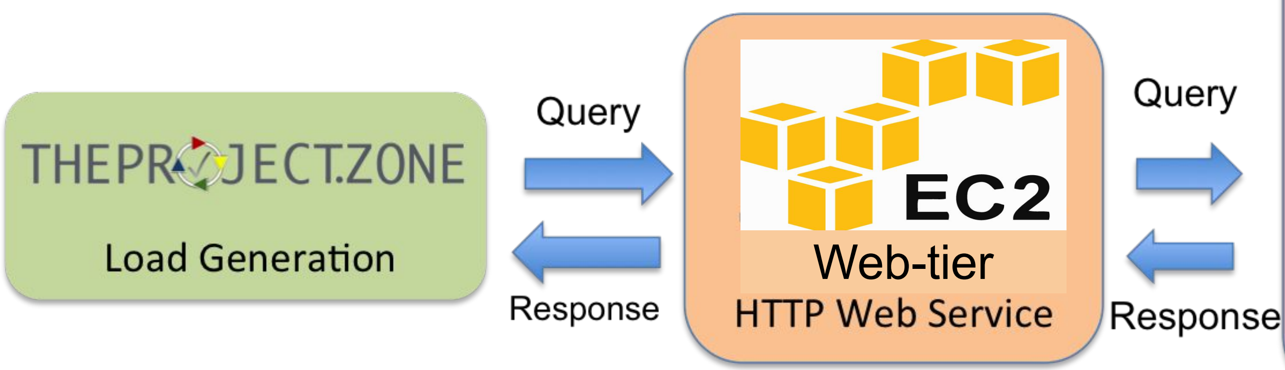
- **Remember to delete the Azure resource group to clean up all the resources in the end**

TEAM PROJECT
Twitter Data Analytics

Team Project

Twitter Analytics Web Service

- Given ~1TB of Twitter data
- Build a performant web service to analyze tweets
- Explore web frameworks
- Explore and optimize database systems



Team Project

- Phase 1:
  - Q1
  - Q2 (MySQL **AND** HBase)

Input your team account ID and GitHub username on TPZ

- Phase 2
  - Q1
  - Q2 & Q3 (MySQL **AND** HBase)

- Phase 3
  - Q1, Q2, & Q3 (Managed Cloud Services)

Query 1 - CloudCoin

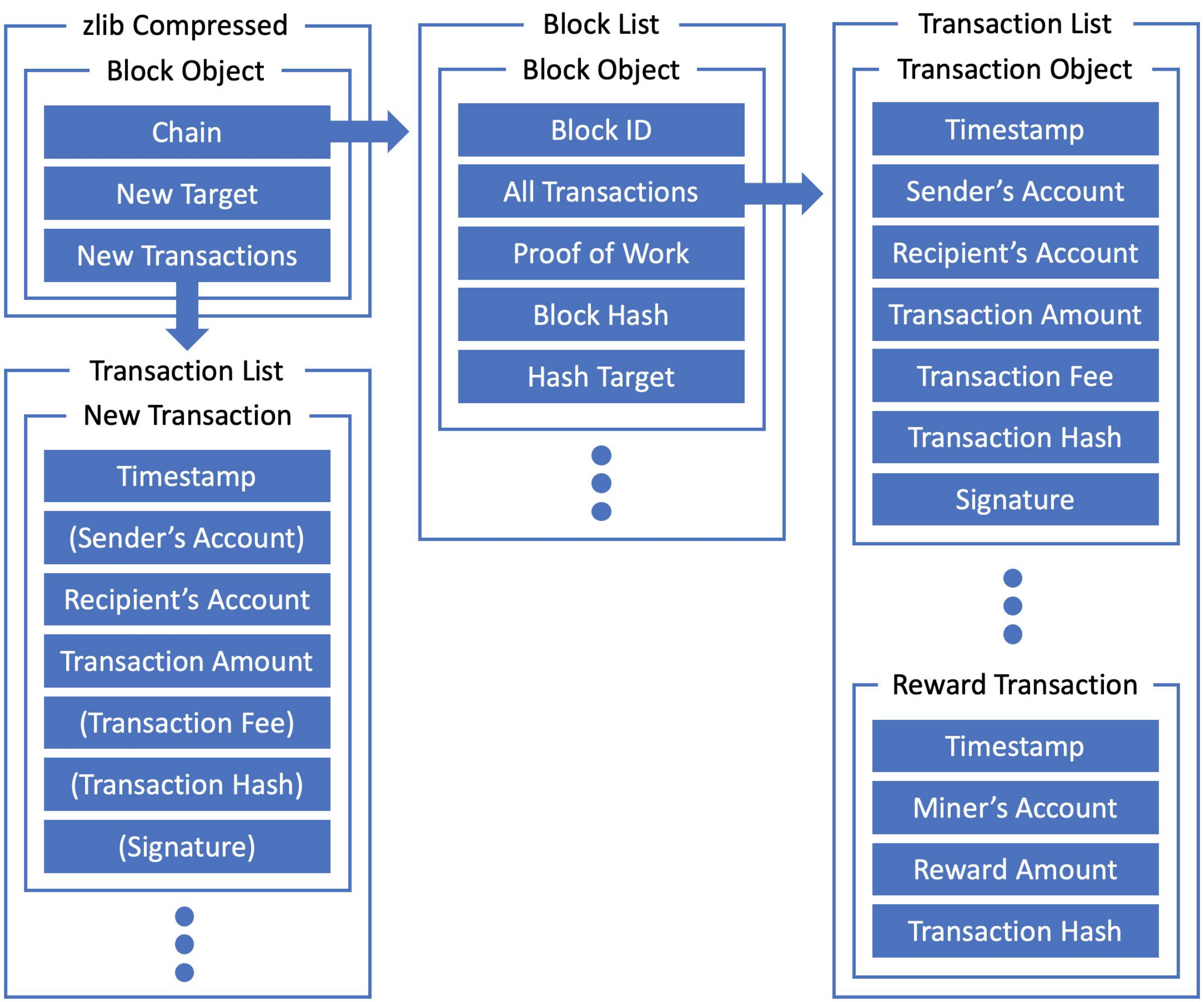
- Query 1 does not require a database (storage tier)
- Implement a web service that verifies and updates blockchains.
- You must explore different web frameworks
  - Get at least 2 different web frameworks working
  - Select the framework with the better performance
  - Provide evidence of your experimentations
  - Read the report first

What is a blockchain, though?

- Data structure that supports digital currency.
- Designed to be untamperable.
- Distributed. Shared among all user nodes.
  - Decentralized
  - Fault Tolerant.
- Consists of chained blocks.
- Each block consists of transactions.

Q1 Example

- **Q1 input:**




```
{
  "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"
    }
  ]
}
```

Q1 Example

```
{
  "all_tx": [...],
  "pow": "cloud",
  "id": 2,
  "hash": "09288a38",
  "target": "0a"
}
```

- **Block:**
  - Created by “miners”.
  - Has a list of transactions.
  - Block hash encapsulates all transaction info and block Metadata, as well as the hash of the previous block, plus a PoW chosen by the miner.
  - Miner finds a PoW (Proof of Work) through brute forcing, to make the block hash lexicographically smaller than the hash target.
  - Block hash formula:

```
CCHash(SHA-256("block_id|previous_block_hash|tx1_hash|tx2_hash|tx3_hash...") + PoW)
```

Q1 Example

- **Transaction:**
  - Signature is computed with hash value using RSA. sig=RSA(hash, key)
  - Hash value computed using all info in the blue box.
  - Transaction hash formula:

```
CCHash("timestamp|sender|recipient|amount|fee")
```


```
{
  "send": 831361201829,
  "recv": 905790126919,
  "amt": 4876921,
  "fee": 78125,
  "time": "1582521009246242025",
  "sig": 829022340937,
  "hash": "46b61f8e"
},
```

Q1 Example

- **Reward:**
  - Special type of transaction.
  - Created by miner.
  - Is the last transaction in the block's transaction list.
  - Reward amount determined by block id, 500000000 for the first two blocks, halved for any two following blocks.


```
{
  "recv": 905790126919,
  "amt": 250000000,
  "time": "1582521603026667063",
  "hash": "b0750555"
}
```

Q1 Example

- **New transactions:**
  - Contains transactions made by your team or by some other accounts.
  - Transaction made by some other account has the same format as any non-reward transaction in the block list.
  - For the transactions made by your team, you need to fill in missing fields and sign it using the key given to you.


```
"new_tx": [
  {
    "sig": 160392705122,
    "recv": 658672873303,
    "fee": 3536,
    "amt": 34263741,
    "time": "1582521636327155516",
    "send": 831361201829,
    "hash": "1fb48c71"
  },
  {
    "recv": 895456882897,
    "amt": 34263741,
    "time": "1582521645744862608"
  }
]
```

Q1 Example

- **Q1 Output:**
  - Collect the new transactions.
  - Create a reward transaction.
  - Include these transactions in a new block.
  - Compute a PoW that makes the new block hash satisfies the new hash target.
  - Append the block to the chain.
  - Respond with the zlib compressed and Base64 encoded new JSON.

Q1 Example

- **Q1 Output:**
  - There will be malicious attempts to break the blockchain.
  - You need to check the validity of the chain.
  - If the chain is not valid, return a string that starts with INVALID.
  - You can append any debug info you want. Just make sure it does not start a new line.
  - E.g., INVALID|any_debug_info_you_like

Query 2 - User Recommendation System

Use Case: When you follow someone on twitter, recommend close friends.

Three Scores:

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

Final Score: Interaction Score * Hashtag Score * Keywords Score

Query:

GET /q2?
user_id=<ID>&
type=<TYPE>&
phrase=<PHRASE>&
hashtag=<HASHTAG>

Response:

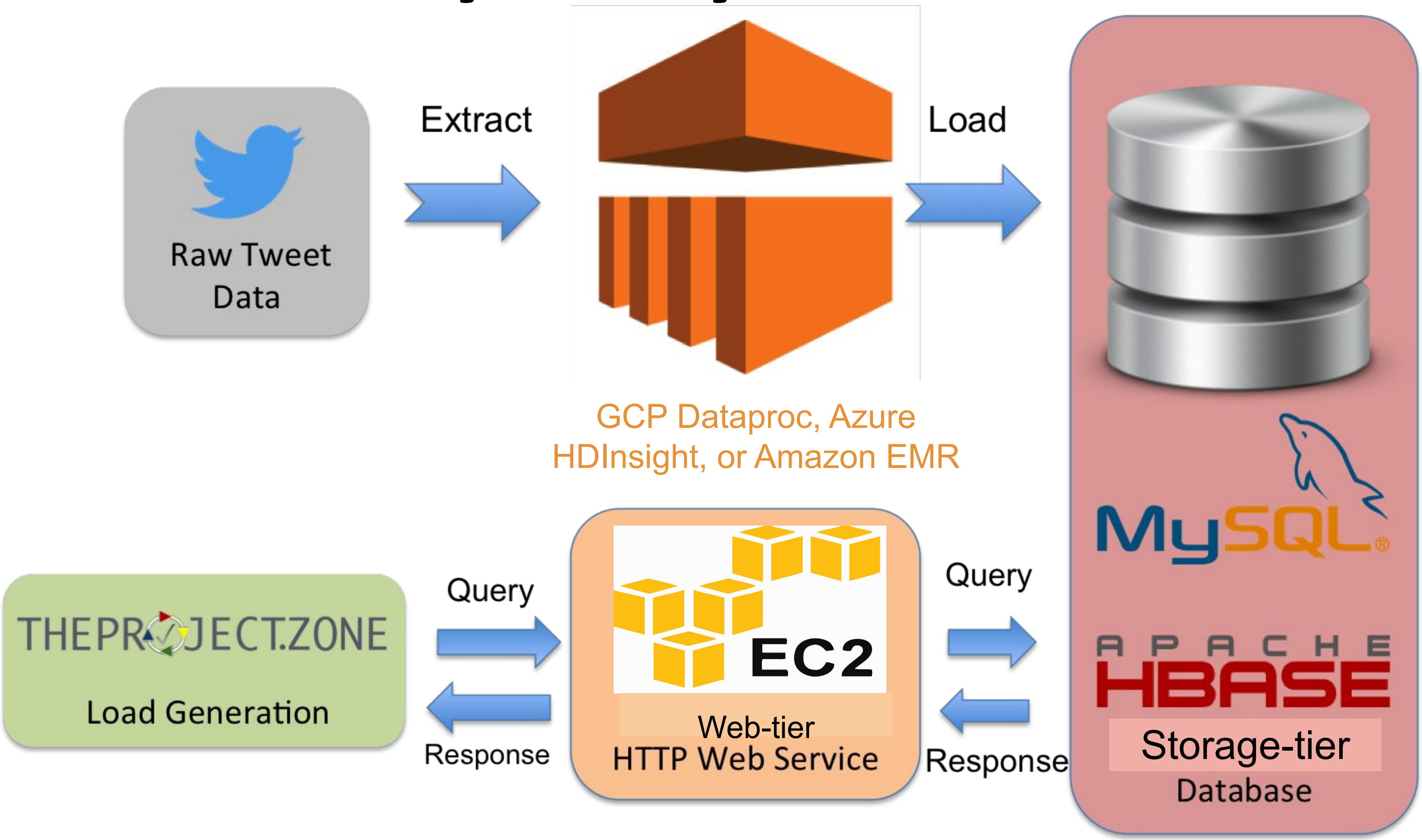
<TEAMNAME>,<AWSID>\n
uid\tname\tdescription\ttweet\n
uid\tname\tdescription\ttweet

Q2 Example

```
GET /q2?
user_id=100123&
type=retweet&
phrase=hello%20cc&
hashtag=cmu
```

```
TeamCoolCloud,1234-0000-0001
100124\tAlan\tScientist\tDo machines think?\n
100125\tKnuth\tprogrammer\thello cc!
```

Twitter Analytics System Architecture



- Web server architectures
- Dealing with large scale real world tweet data
- HBase and MySQL optimization

Git Workflow

- Commit your code to the private repo we set up
  - Update your GitHub username in TPZ!
- Make changes on a new branch
  - Work on this branch, commit as you wish
  - Open a pull request to merge into the master branch
- Code review
  - Someone else needs to review and accept (or reject) your code changes
  - This process will allow you to capture bugs and remain informed on what others are doing

Heartwarming Tips from Your Beloved TAs

1. Design your architecture early and apply for limit increase.
2. EC2 VM is not the only thing that costs money.
3. Primers and individual projects are helpful.
4. You don’t need all your hourly budget to get Q1 target.
5. Coding is the least time consuming part.
6. Think before you do. Esp. for ETL (Azure, GCP, or AWS).
7. Divide workload appropriately. Take up your responsibility.
8. Read the write-up.
9. Read the write-up again.
10. Start early. You cannot make-up the time lost. Lots to finish.
11. I’m not kidding. Drama happens frequently.

Team Project Time Table

Phase	Deadline (11:59PM EST)
Phase 1 (20%) - **Query 1** - **Query 2**	• **Q1 CKPT (5%): Sun, 3/1** • **Report1 (5%): Sun, 3/1** • Q1 FINAL (10%): Sun, 3/8 • Q2 CKPT (10%): Sun, 3/22 • Q2M & Q2H FINAL (50%): Sun, 3/29 • Report2 (20%): Tue, 3/31
Phase 2 (30%) - **Add Query 3**	• Live Test on Sun, 4/12
Phase 3 (50%) - **Managed Services**	• Live Test on Sun, 4/26

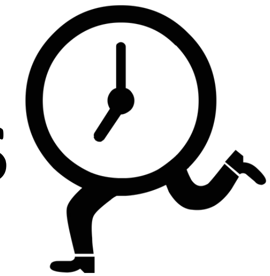
Team Project Deadlines - Phase 1

- Writeup and queries were released on Monday.
- Phase 1 milestones:
  - Q1 Checkpoint: **Sunday, 3/1**
    - A successful 10-min submission for Q1
    - Checkpoint 1 Report
  - Q1 final due: **Sunday, 3/8**
    - Achieve the Q1 target
  - Q2 Checkpoint: **Sunday, 3/22**
    - A successful 10-min submissions:
      - Q2 MySQL **and** Q2 HBase.
  - Q2 final due: **Sunday, 3/29**
    - Achieve the Q2 target for Q2 MySQL **and** Q2 HBase.
  - Phase 1, code and report: **3/31**
- Start early, read the report and earn bonus points!

Suggested Tasks for Phase 1

Phase 1 weeks	Tasks	Deadline
Week 1 ● **2/24**	● **Team meeting** ● **Writeup** ● **Complete Q1 code & achieve correctness** ● **Q2 Schema, think about ETL**	● **Q1 Checkpoint due on 3/1** ● **Checkpoint Report due on 3/1**
Week 2 ● 3/2	● Q1 target reached ● Q2 ETL & Initial schema design completed	● **Q1 final target due on 3/8**
Week 3 ● Spring Break	● Take a break or make progress (up to your team)	
Week 4 ● 3/16	● Achieve correctness for both Q2 MySQL, Q2 HBase & basic throughput	● Q2 MySQL Checkpoint due on 3/22 ● Q2 HBase Checkpoint due on 3/22
Week 5 ● 3/23	● Optimizations to achieve target throughputs for Q2 MySQL and Q2 HBase	● Q2 MySQL final target due on 3/29 ● Q2 HBase final target due on 3/29

This Week's Deadlines



- Quiz 6: OLI Module 13
  Due: Friday, Feb 28th, 2020 11:59PM ET
- Project 3.1: Files v/s Databases
  Due: Sunday, Mar 1st, 2020 11:59PM ET
- Team Project Phase 1 Q1 Checkpoint 1
  Due: Sunday, Mar 1st, 2020 11:59PM ET

Q&A