

# **15-319 / 15-619**

# **Cloud Computing**

## **Weekly Overview 6**

### **P3.1**

March 8, 2021

# Overview

- **Last week's reflection**
  - OLI Unit 3 - Modules 7, 8, 9
  - Quiz 4
  - Project 2.2
- **This week's schedule**
  - Project 2.1 Online Code Review
  - OLI Unit 3 - Module 10, 11, 12
  - Quiz 5
  - OPE - Spark programming
  - Project 3.1
  - Team Project, Phase 1 Q1 Checkpoint

# Reflecting on Last Week

- **Unit 3: Virtualizing Resources for the Cloud**
  - Module 7: Introduction and Motivation
  - Module 8: Virtualization
  - Module 9: Resource Virtualization - CPU
- **Quiz 4**
- **Project 2.2, Containers: Docker and Kubernetes**
  - Docker Intro / Embedded Profile Service
  - Intro to Helm Charts / Deploying MySQL
  - WeCloud Chat Microservices Architecture
    - Autoscaling, Multi-Cloud and Fault Tolerance to Azure

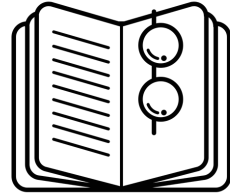
# Activities This Week

- **Unit 3: Virtualizing Resources for the Cloud**
  - Module 10: Resource virtualization (Memory)
  - Module 11: Resource virtualization (I/O devices)
  - Module 12: Case Study
- **Quiz 5**
- **Project 3: Storage and DBs on the cloud**
  - **Project 3.1: Files v/s Databases**
    - Flat files
    - MySQL
    - Redis & Memcached
    - HBase

# Activities This Week, cont.

- **Team Project**
  - **Phase 1** released.
  - **Q1 Checkpoint** due at the end of this week.
    - Query 1 Checkpoint
    - Checkpoint Report
    - Query 1 Early Bird Bonus
- **Project 2.1 Code Review**
  - **Due Date:** Wed, March 10, 23:59:59, ET
  - **Graded activity** which is worth **10** points of P2.1
- **OPE - Spark programming**
  - **Graded**

# This Week: Conceptual Content



- OLI, 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, Unit 3: Modules 10, 11, 12

- Understand two-level page mappings from virtual memory to real pages, from real pages to physical memory
- Learn how memory is overcommitted and reclaimed using ballooning
- Study how I/O requests are intercepted at different interfaces
- Map these concepts into your practical exploration with AWS

# OLI Module 10 - Memory Virtualization

- A process that cannot fit into the physical memory? To run or not to run?
- Page Table
  - Per process
  - Maps virtual addresses to physical addresses
- One level vs. two levels mapping
- Virtual, Real, Physical address spaces
- Memory reclamation
- Ballooning

# OLI Module 11 - I/O Virtualization

- How?
  - Construct a virtual version of the device
  - Virtualize the I/O activity routed to the device
- I/O Basics
- System call interface, device driver interface, and operation-level interface

# OLI Module 12 - AWS Case Study

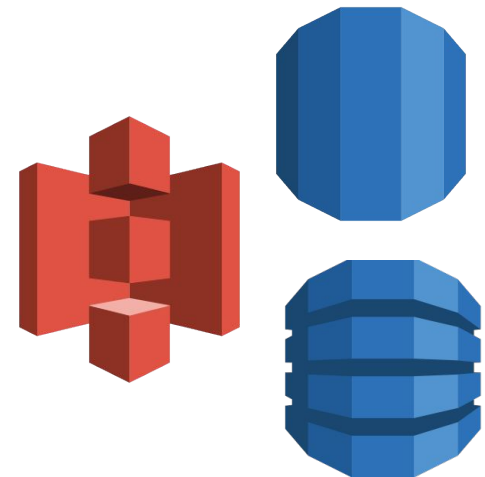
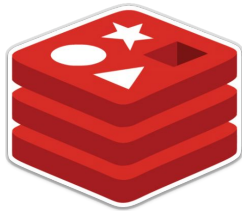
# This Week's Individual Project

- **Project 3: Storage and DBs on the cloud**
  - **P3.1: Files v/s Databases**
    - Comparison and usage of Flat files, RDBMS (MySQL) and NoSQL (Redis, HBase)
  - **P3.3: Replication and Consistency**
    - Multi-threaded Programming and Consistency

# Project 3



mongoDB



# 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
  - **P3.3: Replication and Consistency**
    - Primer: Introduction to Consistency Models
    - Primer: Introduction to multithreaded programming in Java

# MySQL Primer

- **Introduction to *Structured Query Language (SQL)***
  - Data Definition Language (DDL)
    - CREATE, ALTER, DROP
  - Data Manipulation Language (DML)
    - Create: INSERT, Read: SELECT, Update: UPDATE, Delete: 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)**

# 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?

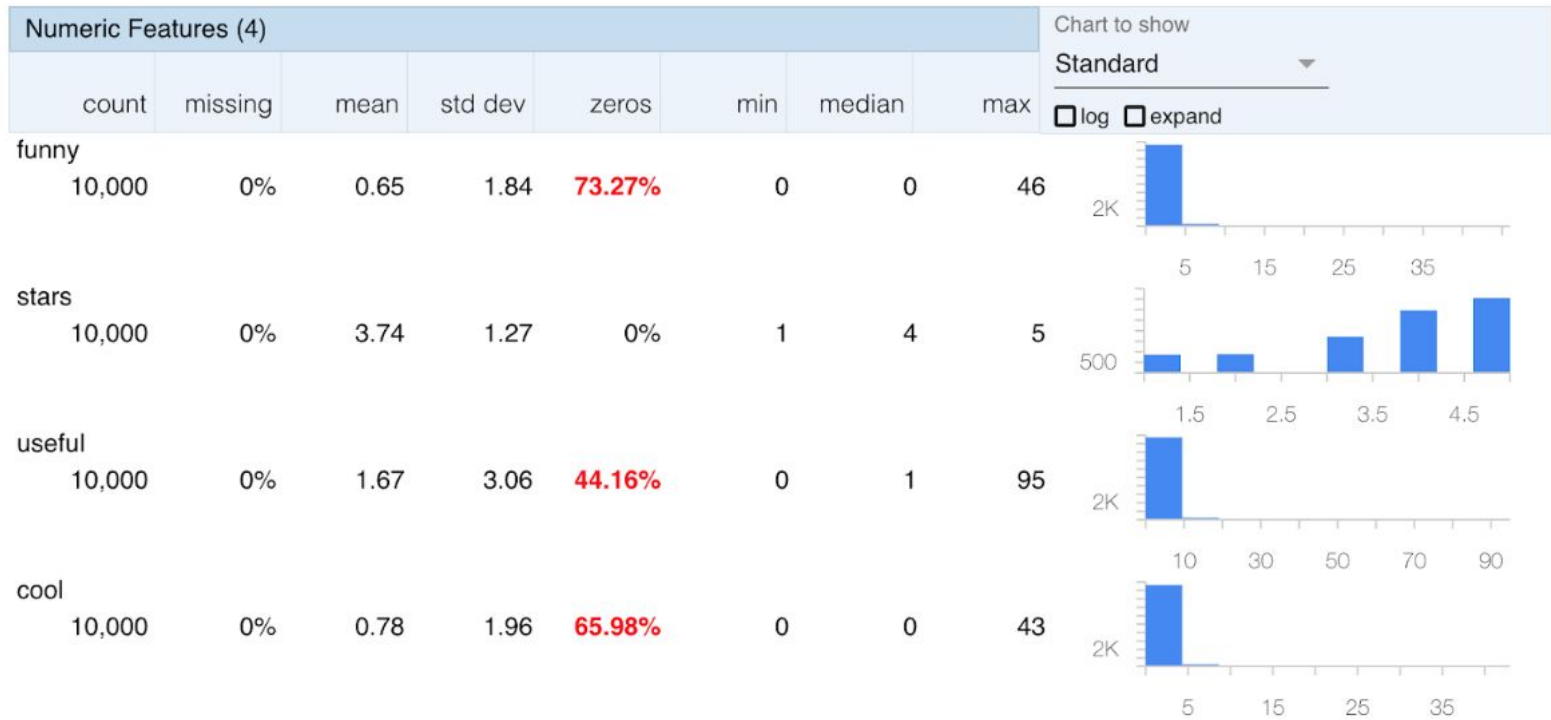
# Dataset

- **Analyze Yelp's Academic Dataset**
- **[https://www.yelp.com/dataset\\_challenge](https://www.yelp.com/dataset_challenge)**
  - business
  - checkin
  - review
  - tip
  - user

# Inspect and visualize data using Facets

Sort by  
Feature order  Reverse order

Features:  int(4)  string(5)



# 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
- **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
- **Productivity**
  - No need to keep switching between your OOP language such as Java/Python, etc. and SQL
- **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

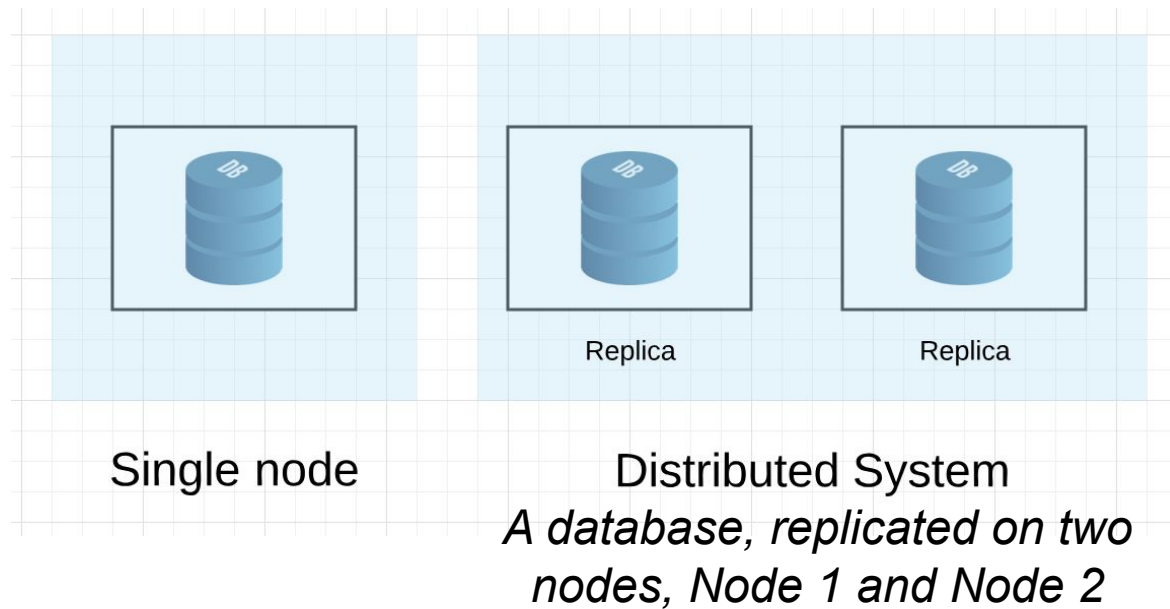
# NoSQL

- **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



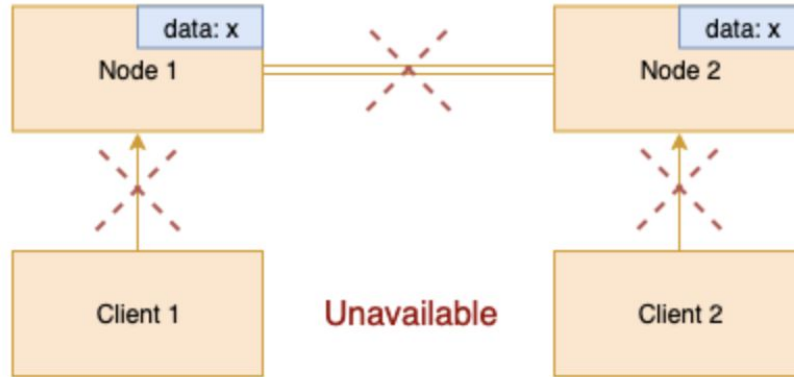
- Since DB is replicated, how is consistency maintained?
- Since the data is replicated, if one replica goes down, will the entire service go down?
- How will the service behave during a network failure?

# CAP Theorem

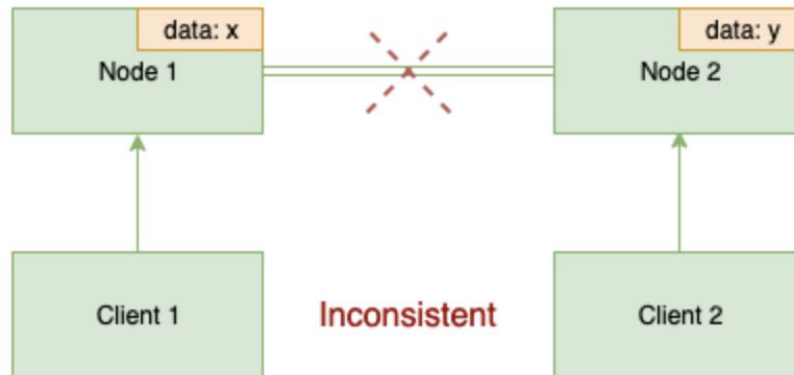
- Only two out of the three are feasible:
  - **CA: non-distributed (MySQL, PostgreSQL)**
    - Traditional databases like MySQL and PostgreSQL have only one server
    - Don't provide partition tolerance
  - **CP: downtime (HBase, MongoDB)**
    - Stop responding if there is partition
    - There will 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

**Consistency & Partition Tolerant (CP)**



**Available & Partition Tolerant (AP)**

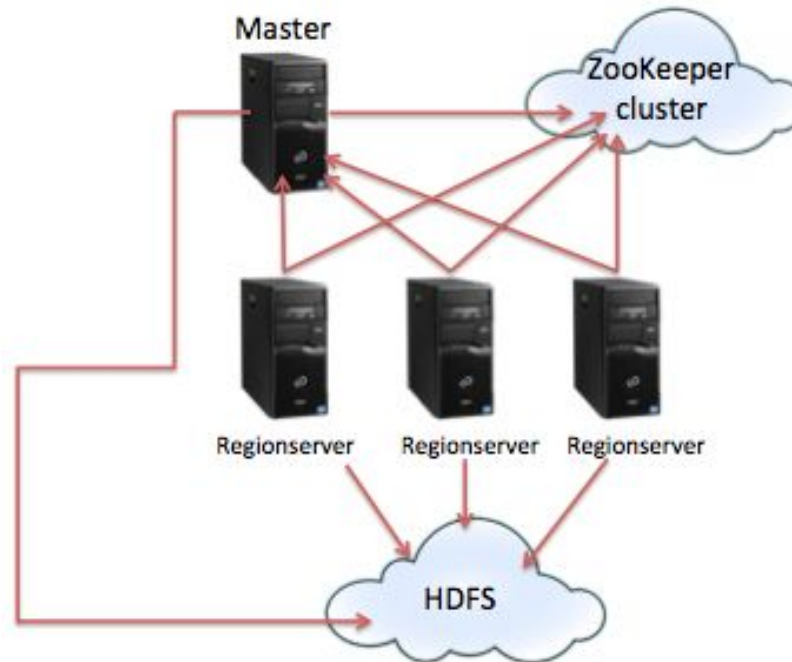


# Task 3: Implement Redis

- 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



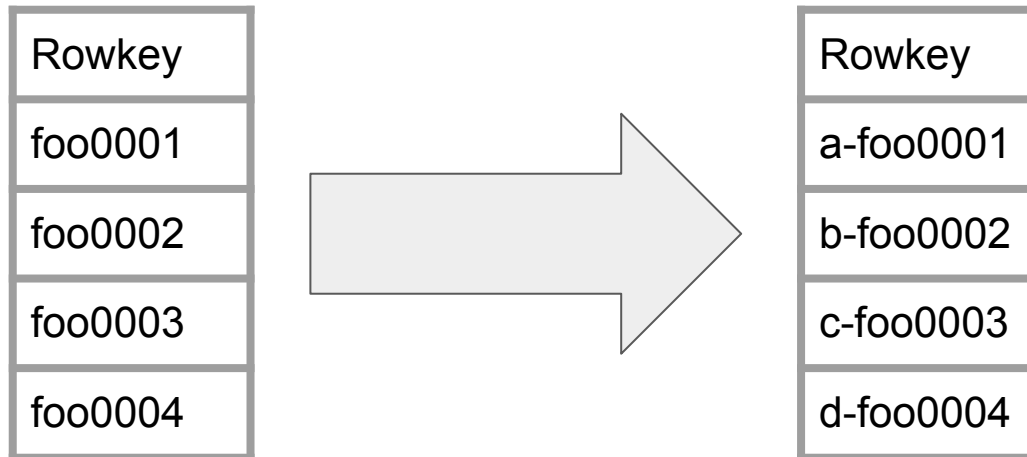
- **Refer to the HBase Basics Primer**

# RowKey Design

- Rows in HBase are sorted lexicographically by rowkey
- **Hotspotting**
  - A large amount of client traffic is directed to one/few node/s
  - The rows are divided into different HRegions
  - Each HRegion contains a contiguous subset of rows
  - HRegionServer is responsible for reading and writing
  - Solution: pre-splitting regions

# RowKey Design - Example 1

Salting: randomly assign prefix



# RowKey Design - Example 1

Salting: randomly assign prefix

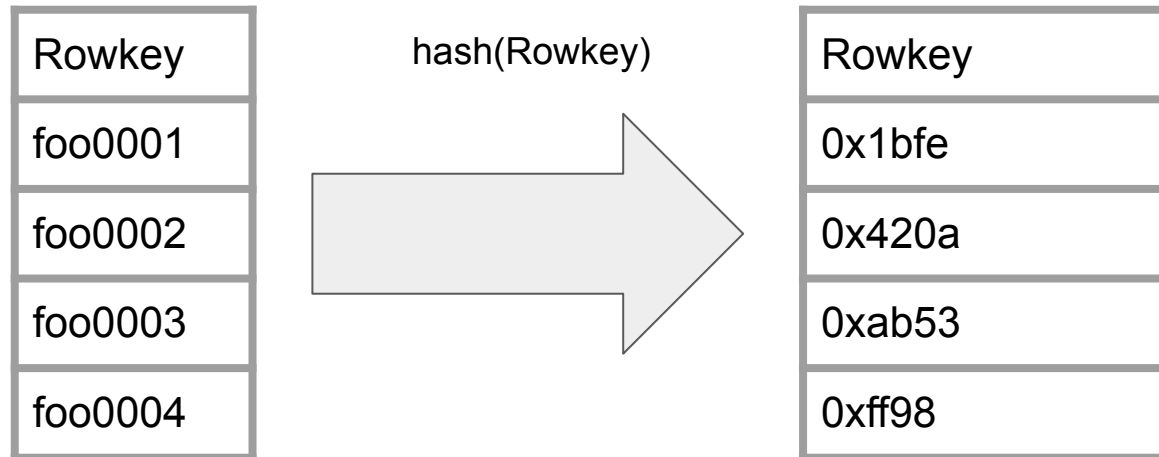
- Command in HBase shell

```
> create 'table', 'example1', SPLITS=> ['b', 'c', 'd']
```

Region	Start Key	End Key	Rows
1		b	a-foo0001
2	b	c	b-foo0002
3	c	d	c-foo0003
4	d		d-foo0004

# RowKey Design - Example 2

## Hashing



# RowKey Design - Example 2

## Hashing

- Command in HBase shell

```
> create 'table', 'example2', {NUMREGIONS => 4, SPLITALGO => 'HexStringSplit'}
```

Region	Start Key	End Key	Rows
1		3fff	0x1bfe
2	3fff	7fff	0x420a
3	7fff	bfff	0xab53
4	bfff		0xff98

# Task 4: Explore HBase

- **Your task:**
  - Launch an HDInsight 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. If you leave an HDInsight cluster running and exceed the budget of the subscription, you will be unable to work on the future Azure projects.**

# Upcoming Deadlines

- Project 2.1 Online Code Review
  - Due on Wednesday, 2021-03-10 23:59 ET
- Quiz 5
  - Due on Friday, 2021-03-12 23:59 ET
- OPE - Spark programming
  - Due on Sunday, 2021-03-14 23:59 ET
- Project 3.1
  - Due on Sunday, 2021-03-14 23:59 ET
- Team Project, Phase 1 Q1 Checkpoint
  - Due on Sunday, 2021-03-14 23:59 ET