Yuhan Tan

(607)–233–3661 - yt628@cornell.edu - linkedin.com/in/yuhan-aaron-tan

SKILLS

Languages: Java, Python, C++, C#, C, SQL, HTML, R
Framework&Tools: Spring, Spring Boot, MySQL, Redis, MyBatis Plus, Kafka, Docker, Linux, Knife4j, .NET Framework, Hadoop, Hive, HDFS, Qt, Git, PyTorch, TensorFlow, Numpy, OpenCV

EDUCATION

Cornell University — **Ithaca, USA**
Candidate for Master of Information Science; GPA: 3.8/4.0 — *08/2023 - Present*

University of Liverpool — **Liverpool, UK**
(First Class Hons)BEng in Computer Science and Electronic Engineering; GPA: 3.8/4.0 — *09/2021 - 07/2023*

Xi'an Jiaotong-Liverpool University — **Suzhou, China**
Major in Computer Science and Technology — *09/2019 - 07/2021*

EXPERIENCE

Software Development Intern | Spring Boot, MyBatis Plus, MySQL, Redis — **05/2024 - 08/2024**
NextTier

- Utilized **Redis** to implement **distributed sessions**, synchronizing login states across clusters. Used **Hash** instead of **String** to store user information, saving memory and facilitating single field modifications.
- Implemented friend similarity matching based on the **edit distance algorithm** to find the most similar users based on tags, employing a **priority queue** to **reduce memory usage** during the TOP N computation.
- **Enhanced concurrency control** by implementing **Redisson distributed locks** to prevent duplicate team joining and exceeding team capacity, ensuring **mutual exclusion** and **API idempotency**.
- **Optimizing caching** by storing user information lists in **Redis**, reducing API response time **from 1 second to 25 milliseconds** and ensuring data integrity with **custom Redis serializers**.
- **Improved initial access speed** by implementing scheduled **cache warming** with **Spring Scheduler**.
- **Enhanced database write operations** by using **custom thread pools** with **CompletableFuture concurrency**, significantly reducing import time for **1 million** rows **from 300 seconds to 54 seconds**.

Software Development Intern | Kafka, Spring Boot — **08/2023 - 12/2023**
Eth Technology

- Developed a **streaming microservice** capable of processing over **1000 events** simultaneously using **Kafka** and **Spring Boot**.
- **Self-learned** Kafka Consumer & Producer patterns **in 1 month**, building **REST API** for event consumption and publication to Kafka topics.
- Implemented **unit tests** and **integration tests** using **JUnit** and **Embedded Kafka**; Conducted **end-to-end manual testing** for different scenarios of data-streaming API using **Postman**; Implemented **concurrency testing** & automated the **load tests** process using **Jmeter**.
- Integrated **Spring JPA** and utilized **H2 database** to store events metadata.

PROJECTS

Asynchronous Processing Framework: AaronFlow | Spring Boot, MySQL, Redis — **03/2024 - Present**
*AaronFlow is an asynchronous task processing framework developed in Java that supports **automatic scheduling, automatic retries**, and **flexible task configuration**.*

- **Architecture Design**: Designed the application with two main layers: Flowsvr (Server) and Aaron (Worker). The Flowsvr layer provides HTTP services for **task querying**, **task scheduling**, and **task management**; the Aaron layer is responsible for **pulling and consuming tasks**.
- **Database Tables Design**: Separated the storage of task information, configuration, and scheduling to reduce dependencies between tables, achieving a **loosely coupled** design. This allows for flexible **task registration** and **management**, and enables quick **task retrieval** through **indexing**.
- **Task Management**: Implemented **timeout task monitoring and recovery** using a **polling** mechanism to regularly check task status, and monitored table size to trigger **table partitioning** logic when thresholds are reached (5 million records).
- **Multi-Worker Optimization**: Initially used **MySQL row-level locking** to prevent multiple Workers from pulling the same batch of tasks, later introduced **Redis distributed locks** from the Worker side.
- **Performance Optimization**: Conducted **stress testing** and analyzed performance bottleneck. By using a **MySQL connection pool** and increasing the maximum number of connections, improved performance **from 100 QPS to 500 QPS**.