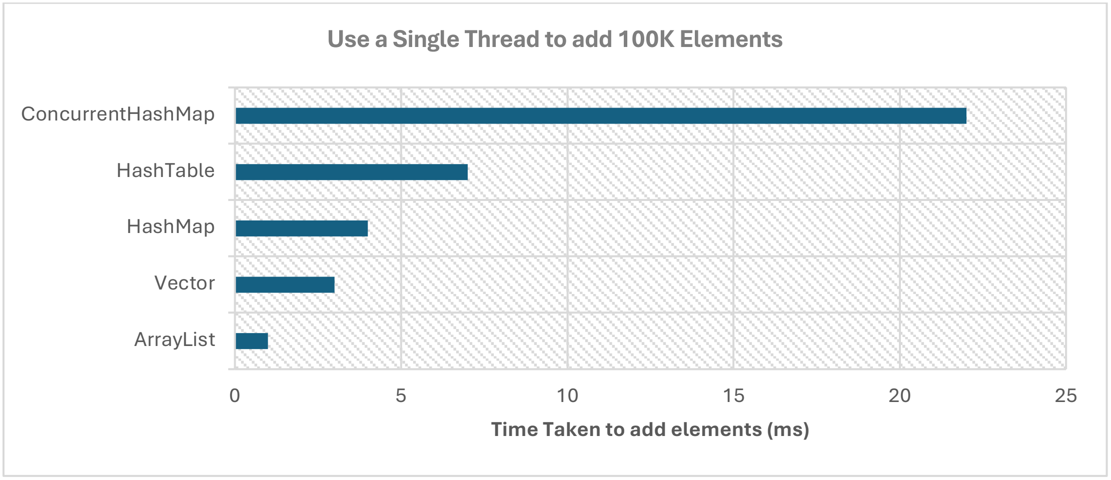
Midterm Report

Before taking this course, I had some knowledge of building an application, primarily focusing on front-end design and the connection between the front end and back end. However, after taking this course, I have gained a much deeper understanding of the details involved in building a durable system, as well as the strategic aspects of system design, particularly in cloud server (notably AWS) deployment, configurations, and scaling.

What I've Learned So Far

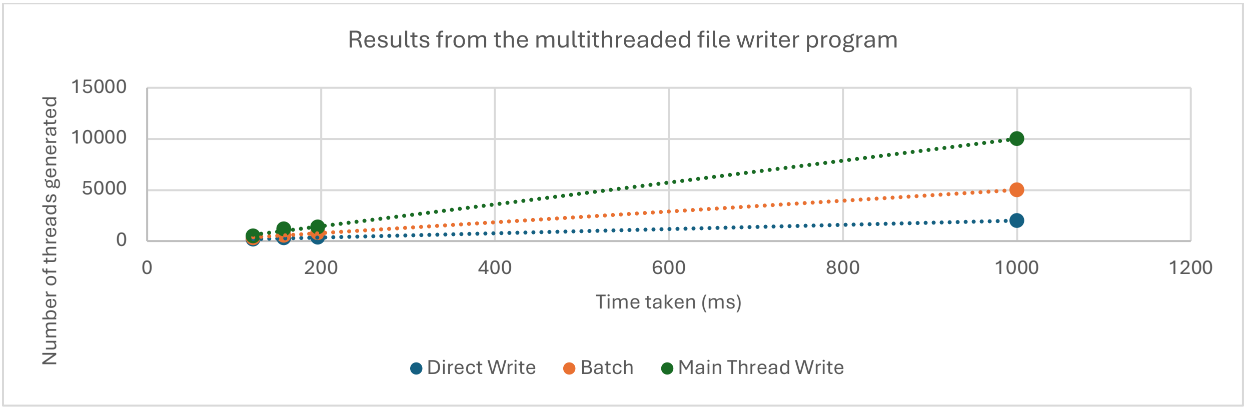
At the very beginning, I deepened my understanding of **data structures** such as vectors, array lists, hash tables, hash maps, and ConcurrentHashMap. While I was previously familiar with their theoretical representations, the homework assignments pushed me to explore their practical usage in real-world applications.

- Vectors and array lists are both dynamic arrays, but vectors are synchronized, making them thread-safe at the cost of performance. Array lists, on the other hand, are not synchronized and offer better performance in single-threaded scenarios.
- Hash tables and hash maps both provide key-value storage, but hash tables are synchronized, ensuring thread safety at the expense of speed. Hash maps, being unsynchronized, offer better performance in non-concurrent environments. ConcurrentHashMap, however, improves upon hash maps by providing efficient thread-safe operations through segment-based locking, making it more suitable for multi-threaded environments.
- Result from 2a, a simple experiment on understanding data structures' impacts on performance



I have witnessed firsthand how different data structures impact program performance, helping me make better design choices.

I also witnessed the impacts of performance by different file writer programs, specifically, direct write, bath and main thread write. I learned that batch write was the fastest in terms of performance as it reduces the overhead of frequent I/O operations and synchronization overhead; and it is also the most scalable approach as it writes data in batches after all threads have completed their tasks. Below is a graph on a simple experiment on the subject:

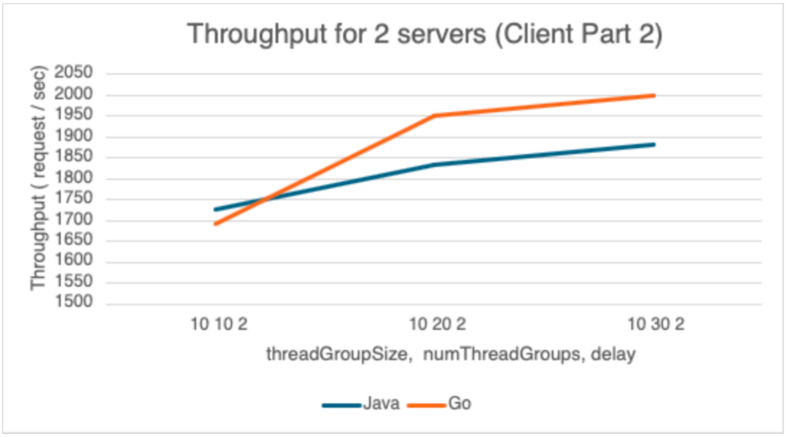


Later, I learned about **AWS EC2**, which is one of the biggest areas of my knowledge growth. While I was aware of its existence before, I now have a thorough understanding of its configurations, including security and inbound rules, Elastic IPs, key pairs, load balancers, snapshots, and hosts. Thanks to the hands-on homework assignments, I have gone beyond just having a vague understanding and have gained real experience working with each of these components. For example, instead of knowing that load balancers help distribute incoming traffic across multiple instances to enhance performance and fault tolerance, I am able to configure the target groups to define which instances will receive traffic; instead of knowing the key concept of Elastic Ips, I am able to associate elastic Ips with instances to maintain a consistent IP address for instances for easier access to instances.

I also learned extensively about **Tomcat**, not only in terms of deploying Java servlets and managing functions through the Tomcat Manager dashboard but also in its deeper configurations. I now understand important settings like maxThreads (defines the maximum number of simultaneous requests it can handle, affecting concurrency and system performance) and maxConnects (number of simultaneous connections that can be maintained), where Tomcat stores system logs, and how it controls access. This knowledge has helped me better configure and optimize Tomcat for different use cases.

Besides configurations and data structures, my experience with **Go** has rapidly deepened. Before the course, I had only written a simple Go server once. Now, I am becoming increasingly comfortable with the Go language and goroutines, and I am beginning to truly appreciate its simplicity when it comes to networking and web development. Especially, Goroutines are lightweight threads managed by the Go runtime, enabling efficient concurrency. The ease of handling networking in Go, such as setting up HTTP servers with the `net/http` package, makes it an excellent choice for web development. Writing concurrent programs in Go has given me a deeper appreciation for its power and efficiency.

Below is a graph on Go’s excellent performance over Java when using Goroutine from homework 4a:



Throughout the process, I have started to distinguish and internalize key computing concepts such as **multithreading**, **concurrency**, and **parallelism**. Multithreading involves running multiple threads within the same application, enabling efficient CPU utilization. Concurrency allows multiple tasks to progress independently, while parallelism involves executing multiple tasks simultaneously across CPU cores. Nodes refer to individual computing units in a distributed system, while servers handle client requests and distribute workloads efficiently. Understanding these concepts has been crucial in building scalable distributed systems.

Libraries have been another major area of growth, particularly with **Apache HTTP Client** and **Apache Spark**. Apache HTTP Client is a library used for making HTTP requests in Java, providing capabilities like connection pooling, timeouts, and authentication mechanisms. Apache Spark is a distributed computing framework used for big data processing. Working with these libraries has given me a better understanding of how they can be leveraged to build efficient and scalable distributed systems.

In addition to Tomcat, I have also learned about **RabbitMQ** and **RPC**. RabbitMQ is a message broker that facilitates asynchronous communication between microservices using a queuing system. It enables decoupled architecture, where different parts of the system communicate without direct dependencies. RPC allows one program to execute functions on a remote system as if they were local, enabling seamless distributed computing. Understanding how messaging queues and remote calls work has broadened my perspective on system design and communication between different services.

Furthermore, I have learned about **Maven**, **Jconsole**, **AWS Lambda**, **Packer**, **Terraform**, **AMI**, **RDS**. Maven is a build automation and dependency management tool for Java projects, which I have heavily relied on to efficiently manage project dependencies, build lifecycle phases, and integrate plugins to streamline software development. JConsole is a monitoring app on Java applications, including tracking memory usage, CPU consumption, which has been instrumental in understanding and optimizing performance. AWS Lambda is a serverless computing service that allows execution of functions in response to events without managing infrastructure. Packer automates the creation of machine images, enabling consistent environment replication. Terraform is an infrastructure-as-code tool that defines and provisions resources programmatically. AMI contains pre-configured system settings used for launching EC2 instances. RDS provides managed databases with automatic scaling and backups. These tools have introduced me to infrastructure as code and automated deployments, making it easier to manage scalable cloud infrastructure.

One concept that I will likely remember forever is **Little's Law**. Running numerous tests and experiments with different variables has solidified my understanding of how request rates, service time, and concurrency levels affect system performance. These experiments have given me valuable insights into system behavior under different loads.

What I Learned & Contributed to the Team

Furthermore, the collaboration with my teammates is one of the most valuable aspects of this course. Through our hands-on projects and discussions, we supported each other throughout the learning process. We brainstormed solutions to complex problems, engaged in fantastic discussions on debugging, optimization strategies, and system design choices. We examined where things could be wrong or improved, discussed unexpected metrics, and analyzed potential causes.

In the two milestones, I contributed the most by working on building the server and client, setting up configurations, and ensuring everything functioned correctly. I worked around different system setups and optimizations to make our application run efficiently. Additionally, I actively participated in discussions on the metrics we obtained, identifying areas of improvement, and optimizing performance based on test results. Apart from the technical contributions, I also played a role in team management, ensuring smooth coordination, assigning tasks, and maintaining effective communication among team members.

What I Planned for the Future

As I continue to build upon the knowledge gained in this course, I have identified several areas that I plan to explore further. Some of these topics were introduced in our homework assignments but left open-ended, presenting opportunities for deeper investigation.

In Homework 4a, where our team focused on *Maximizing Throughput in Distributed Systems using Java & Go Servers alongside a Java Client*, we made significant progress in optimizing performance. However, we did not fully uncover why using *runnable* tasks for the client resulted in better performance than *un-runnable* ones. Thus, I plan to explore further on:

- Client-side: if older, long-running client threads affect the performance of newer requests, potentially leading to bottlenecks or uneven request distribution?
- Server-side: if server scaling strategies, particularly with Kubernetes, would help improve performance significantly?

In Homework 6a, where my team worked on *Experimenting and Optimizing Servers with AWS RDS and ALB*, we focused on fine-tuning cloud-based deployments. Due to time-constraints, we could not experiment all of our ideas, afterwards, I plan to keep investigating on methods to improve performance:

- Client-side: if adding a Read-Only Replica could greatly reduce latency?
- Server-side: if upgrading AWS EC2 instances from t3.micro to t3.medium (or higher) could greatly reduce loads on CPU?

These experiments will help refine my understanding of distributed systems performance at both the client and server levels.

What I Summarized & Reflected

All the knowledge I have gained in this course interweaves, reinforcing and supporting each new concept learned. No single piece of knowledge stands alone; instead, everything I have learned so far continuously builds upon itself. I keep revisiting and deepening my understanding of each concept, refining my ability to apply these ideas in practice. This iterative process ensures that I not only retain what I have learned but also improve my skills and adaptability in various scenarios.

Last but not the least, the most important lesson I have learned in this course is **how to troubleshoot and solve problems independently**. I have learned to be patient – again and again. Dealing with frustrations and overcoming emotions has been a critical skill, and I now understand that calming myself is the first step toward solving any problem. This ability to methodically debug and resolve issues has been one of the most valuable takeaways from the course.

Overall, this course has significantly expanded my technical knowledge and problem-solving skills. I feel more confident in designing and deploying distributed systems, and I have developed a deeper appreciation for the complexities involved in building scalable and reliable cloud-based applications. I am excited for the second half of the course.