Towards a Self-Healing Runtime for Go

Sai Vamsi Alisetti
Perm # - A1U0061

Abhishek Kumar
Perm # - A3F8A76

Vision Statement

Concurrency is a core feature of the Go programming language, enabling developers to efficiently utilize multi-core processors. However, common concurrency issues such as deadlocks, race conditions, and goroutine leaks can severely impact system reliability. The goal of this project is to first develop a deep understanding of Go's concurrency model, benchmark different concurrency mechanisms, and ultimately build a self-healing runtime tool capable of detecting and recovering from concurrency issues dynamically.

Goals & Objectives

- Gain proficiency in Go's concurrency model, including goroutines, channels, and synchronization primitives.
- Benchmark the performance of different concurrency mechanisms (e.g., mutexes, channels, atomic operations).
- Develop an automated system to detect and recover from concurrency issues like deadlocks, race conditions, and goroutine leaks.
- Evaluate the effectiveness of the self-healing runtime through empirical analysis.

Project Plan

1. **Phase 1**: Learn Go Programming Language.
2. **Phase 2**: Learning Go's Concurrency Model.
    - Goroutines
    - Channels (buffered and unbuffered)
    - Implement simple programs to understand common pitfalls such as race conditions, deadlocks, and starvation.
3. **Phase 3:** Benchmarking Go's Concurrency Mechanisms
    - Develop a benchmarking suite
    - Goroutine scheduling behavior under high contention
    - Impact of goroutine leaks on memory usage
4. **Phase 4:** Designing a Self-Healing Runtime
    - Deadlock Detection & Recovery
    - Use timeouts and forced unlocking strategies to resolve deadlocks
    - Goroutine Leak Detection
    - Implement the self-healing runtime as a Go library.

Sai Vamsi Alisetti

Abhishek Kumar