Implementing Distributed Garbage Collection in a Microservice-Based System

Sai Vamsi Alisetti
Perm # - A1U0061

Abhishek Kumar
Perm # - A3F8A76

Vision Statement

The goal of this project is to implement an efficient and fault-tolerant Distributed Garbage Collection (GC) system for a microservices-based environment. The system will handle the reclamation of unused memory and resources across distributed nodes (or services) to optimize system performance and prevent memory leaks.

Background

Distributed systems, especially microservices, can lead to complex memory management challenges due to the decentralized nature of resources. In such systems, garbage collection (GC) must be carefully designed to ensure the following:

- Efficient use of system memory.
- Minimal disruption to running services.
- Fault tolerance and system resilience.

This project aims to evaluate different distributed garbage collection strategies and implement a system that can manage memory across nodes while addressing these challenges.

Goal and Objectives

- A functional, scalable distributed garbage collection system with a clear understanding of the strengths and weaknesses of each strategy in the context of a microservices architecture.
- Detailed performance benchmarks that can guide the selection of the best garbage collection strategy for specific use cases.

Project Plan

- Research & Design:
  - Study Reference Counting, Mark-and-Sweep, Epoch-based GC, and Distributed Tracing.
  - Define system architecture (e.g., microservices, messaging protocols).
  - Choose tools & technologies (Python/Go, gRPC/Kafka, Docker/Kubernetes).
- Prototype Development:
  - Implement a basic distributed GC system.
  - Set up inter-node communication for reference tracking.
  - Perform initial testing for correctness and performance.
- Implement Additional Strategies:

  - Extend the system to include Mark-and-Sweep and Epoch-based GC.
  - Optimize synchronization and memory management techniques.
  - Begin benchmarking for latency, memory usage, and overhead.
- Evaluation & Finalization:
  - Compare all implemented strategies using test workloads.
  - Identify trade-offs in performance, scalability, and fault tolerance.
  - Document findings, challenges, and potential improvements.

Technologies

- Programming Languages: Python, Go, or Java
- Microservices Orchestration: Kubernetes (K3s)
- Inter-Service Communication: gRPC, Kafka
- Monitoring & Logging: Prometheus, Grafana, ELK Stack
- Benchmarking Tools: Locust, JMeter, Custom Profiling Scripts

Sai Vamsi Alisetti

Abhishek Kumar