Load Balancer Hashing

Daniela Gallegos Dupuis

Due February 21th, 2025

SENG 468 Software Scalability

Table of Contents

Part 1: Server Failure in Normal Hashing.. 1

(a) Impact of a Server Failure (2 Marks)... 1

(b) Recalculating New Assignments Using Normal Hashing (4 Marks)...................................1

(c) Disadvantages of Normal Hashing (4 Marks)... 1

Part 2: Using Consistent Hashing (10 Marks)..3

(a) Implementation of Consistent Hashing Rules (3 Marks).. 3

(b) Comparison with Normal Hashing (3 Marks)...3

(c) Benefits of Consistent Hashing (4 Marks)... 3

Part 1: Server Failure in Normal Hashing

(a) Impact of a Server Failure

When Server S4 (handling hash positions 890-999) fails, all student grades stored on it become inaccessible, including Student ID 404. Without redundancy, the affected data remains unavailable until reassigned, causing system downtime as records are redistributed. The server receiving the reassigned data may experience increased workload, leading to inefficiencies. Additionally, cached data on the failed server is lost, requiring caches to be rebuilt, which increases query latency.

(b) Recalculating New Assignments Using Normal Hashing

In normal hashing, the assigned server is determined using the modulo operation. When Server S4 fails, the affected key, Student ID 404, is reassigned to S0 since 404 mod 5 = 4. With only one key affected, only one reassignment was needed. However, server failure increases retrieval latency as data must be accessed from a different server. Redistributing data can also create an uneven workload, potentially overloading certain servers. If multiple servers fail, large-scale reallocation is required, making recovery slow and inefficient.

(c) Disadvantages of Normal Hashing

Normal hashing has several disadvantages, particularly in handling server failures. When a server fails, all its data must be reassigned, increasing computational overhead and causing cache misses as caches need to be rebuilt. Load balancing issues arise as some servers may receive a disproportionate number of reassigned records, leading to inefficiencies. Scalability is challenging because adding or removing servers triggers widespread key redistribution, increasing computational overhead and system latency.

Part 2: Using Consistent Hashing

(a) Implementation of Consistent Hashing Rules

Consistent hashing offers a more efficient approach than normal hashing by using a circular hash ring, where each server is assigned multiple positions through virtual nodes. When a server fails, only its affected keys are reassigned to the next available server in a clockwise direction, minimizing disruption. In the execution results, the failure of Server S2 resulted in the reassignment of Student ID 505 to Server S1. This targeted redistribution ensures that only necessary keys are moved, avoiding large-scale data reallocation and maintaining system stability.

(b) Comparison with Normal Hashing

Consistent hashing minimizes key reassignment during server failures, unlike normal hashing, which redistributes all affected keys and causes major disruptions. Instead, only the keys mapped to the failed server are reassigned to the next available server, reducing unnecessary data movement. This efficiency is achieved through a circular hash space, where reallocation is confined to adjacent nodes, maintaining system stability and minimizing downtime.

(c) Benefits of Consistent Hashing

Consistent hashing offers key advantages over normal hashing, making it the preferred choice for distributed systems. It enhances scalability by allowing new servers to be added with minimal key redistribution, enabling dynamic system growth without major reconfiguration. Fault tolerance is improved as failures affect only a small subset of records, preserving system stability and ensuring continued operation. Caching efficiency is also maintained, as fewer records are reassigned, reducing cache misses and improving lookup speeds. Due to its ability to balance workloads, minimize disruptions, and ensure high availability, consistent hashing is widely used in cloud storage, content distribution networks, and distributed databases.