Scalability Analysis Report

Introduction

This report presents a scalability analysis of a concurrent hash table implementation. The analysis evaluates the performance of the hash table under varying loads and concurrency levels.

Performance Metrics

- **Throughput**: The number of operations the system can handle per unit of time (operations per second).
- **Latency**: The time it takes to complete a single operation (seconds per operation).

Techniques Used

To achieve efficient concurrency and synchronization, the following techniques were employed:

- **Readers-Writer Locks (Read-Write Locks):** These locks allow multiple threads to read data concurrently while ensuring exclusive access for write operations. Previously I've implemented a single readers-writer lock for the entire table, now I improve the granularity, by moving the lock to each buckets.
- **POSIX Threads (pthreads):** POSIX threads provide a standardized API for creating and managing threads in a Unix-like environment. Using pthreads, the implementation supports multi-threaded access to the hash table, enabling parallel processing of operations.
- **Circular buffer:** With this technique the communication between server and client is not dependent anymore to a flag (that previously indicates if the server was ready for a new command), but now the client writes its request on this buffer, and a tail pointer takes track about the position. Server uses a head pointer to check if there are new commands written on the circular buffer, and elaborate the request.
- **Shared memory:** I create a shared memory between server and client, in order to print the server output.

Test Scenarios

1. **Varying the Number of Threads**:
    - Test with 2, 4, 8, and 16 threads.
    - Each thread performs a fixed number of operations.
2. **Varying the Number of Operations per Thread**:
    - Test with 50,000, 100,000, 200,000, and 400,000 operations per thread.
    - Use a fixed number of threads.

Experimental Setup

1. **Hardware**: Tests were conducted both on an MacOS and Linux based operating system.
2. **Software**: The hash table implementation uses POSIX threads for concurrency and readers-writer locks for synchronization.

Results

Number of thread	Operations per thread	Total operations	Time taken (seconds)	Throughput (operations/second)	Latency (operations/microsecond)
4	10	40	0,0032	1,25E+04	80
4	50	200	0,0097	2,06E+04	48,5
4	80	320	0,0179	1,79E+04	55,9375
4	100	400	0,0184	2,17E+04	46
4	200	800	0,0382	2,09E+04	47,75
4	400	1600	0,0786	2,04E+04	49,125

Analysis

The hashtable program demonstrates efficient handling of increasing operations and threads, maintaining relatively constant latency (execpt for the first use case, the others are around 50 operaitons over microsecond) and high throughput.

Conclusion

The concurrent hash table implementation demonstrates good scalability, handling increased loads and concurrency levels effectively. The system maintains high throughput and low latency, making it suitable for high-performance applications requiring efficient concurrent data access.