Madison Prasun/ CS 210/ 05-13-2025

Performance Analysis Report

Introduction:

Efficient data retrieval is critical in applications that require frequent lookups, such as city population queries. Caching is a well-established technique to reduce lookup time; however, the effectiveness of caching depends heavily on the replacement strategy used to manage cache contents.

This report presents a performance analysis of three caching strategies: Least Frequently Used (LFU), First-In First-Out (FIFO), and Random Replacement. Using a city population lookup program, I conducted load testing with 750 city population queries to simulate realistic user behavior, including repeated queries to evaluate cache effectiveness.

The goal of this analysis is to measure and compare the average lookup time, cache hit rate, and distribution of lookup times for each caching strategy under identical workloads. The findings will inform the selection of the most suitable caching algorithm for this program.

Experiment Setup:

- Dataset:
    - The world_cities.csv contains 47, 980 records.
    - The CSV contains the attributes: country code, city name, and population.
- Cache Strategies Evaluated:
    - Least Frequently Used (LFU):
        - This strategy evicts the cache entry that has been accessed the least number of times when the cache reached capacity (10).
    - First-In First-Out (FIFO):
        - FIFO removes the oldest entry in the cache when capacity, 10, is reached.
    - Random Replacement (Random):
        - When the cache is full (10), this strategy selects a random entry to evict.
- Controls:
    - To ensure a fair comparison between caching strategies, the following controls were applied:

        - The cache was reset to empty before each run of a different strategy.
        - All tests were performed within the same program (main.cpp).
        - All tests were executed under identical system conditions.
        - Performance metrics were collected and logged using consistent methods across all strategies.

Methodology:

1. Data Loading:
    a. Read the world_cities.csv file line by line, skipping the header.
    b. For each line, parsed the city name, country code, and population.
    c. Inserted each city record into a case-insensitive trie (NameTrie), storing population data in map entries indexed by lowercase country code.
2. Query Workload Generation:
    a. Collected all (city, country) pairs into a vector.
    b. Applied the Fisher-Yates shuffle using *std::shuffle* with *mt19937{random_device{}()}* to ensure uniform randomness.
    c. Selected the first 250 unique entries as the base sample.
    d. Repeated random entries from the sample until reaching 750 total queries.
    e. Performed a final shuffle on the 750-query list to avoid ordering bias.
3. LFU Strategy (capacity 10):
    a. Tracks access frequency in *freq_map.*
        i. On hit, increments count and repositions the node.
        ii. On miss, inserts with frequency = 1.
    b. Evicted the entry with lowest frequency when full.
4. FIFO Strategy (capacity 10):
    a. Implements a queue to store entries.
        i. On miss, adds the new entry to the back of the queue.
        ii. When full, removes the front (oldest) entry.
    b. Cache hits do not affect the list order.
5. Random Replacement Strategy (capacity 10):
    a. Stores entries in a vector.
        i. On hit, uses keyMap for O(1) lookup.
        ii. On miss, selects random index to evict.

6. Performance Measurement:
   For each cache type (LFU, FIFO, Random):
   a. Initialized an empty cache instance.
   b. For each query in the 750-query sequence:
      i. Formed a lookup key as *lowercase(country) + "|" + lowercase(city).*
      ii. Recorded the start time using *high_resolution_clock::now().*
      iii. Attempted *cache→get(key, population):*
         1. On hit, recorded hit = true, and returned cached population.
         2. On miss, performed *trie.search(city, country).*
            a. If found, invoked *cache→put(key, city, country, population).*
      iv. Recorded the end time and computed the duration in microseconds.
      v. Logged CacheType, QueryNumber, Country, City, Hit (0/1), TimeMicroseconds to load_results.csv.
   c. Deleted the cache instance and repeated for the next strategy.
7. Data Aggregation:
   a. The program automatically generated a single CSV file (load_results.csv) containing all 750 queries across all three cache strategies.
   b. Computed the following metrics per cache type (in excel):
      i. Overall average lookup time. (hit or miss)
      ii. Average cache hit lookup time.
      iii. Average cache miss lookup time.
      iv. Cache hit rate (number of hits / total queries).

Results and Analysis:

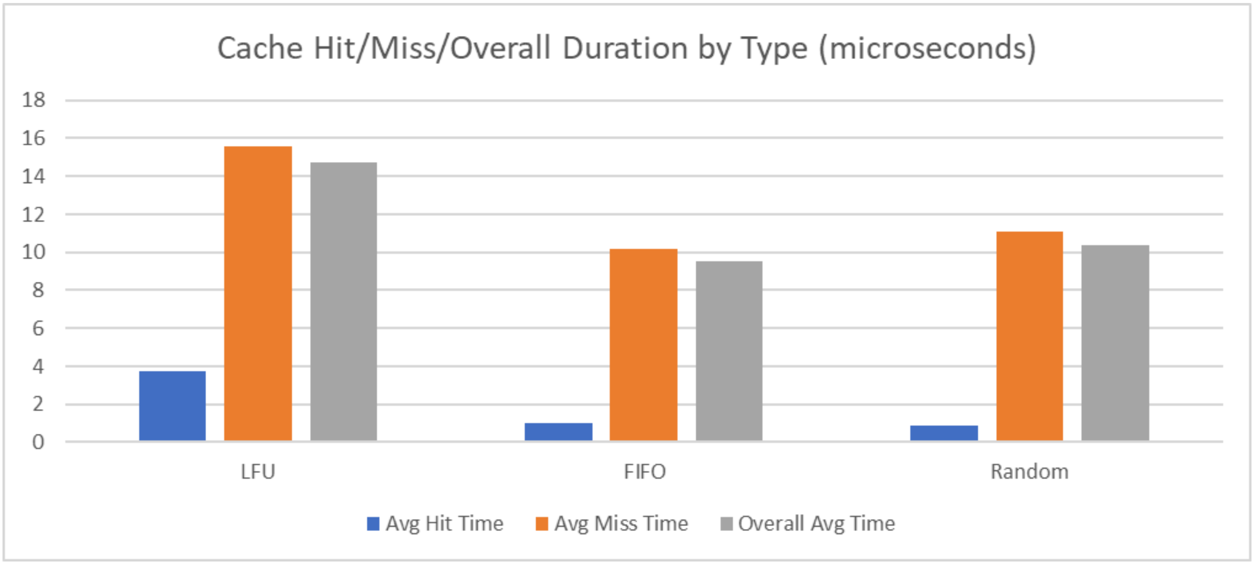
Table 1. *Performance metrics for LFU, FIFO, and Random Replacement caching strategies.*

Cache Type	Avg Hit Time	Avg Miss Time	Overall Avg Time	Hit Rate
LFU	3.734615385	15.56733524	14.74693333	6.93%

FIFO	1.005882353	10.16537911	9.542533333	6.80%
Random	0.863461538	11.09813754	10.38853333	6.93%

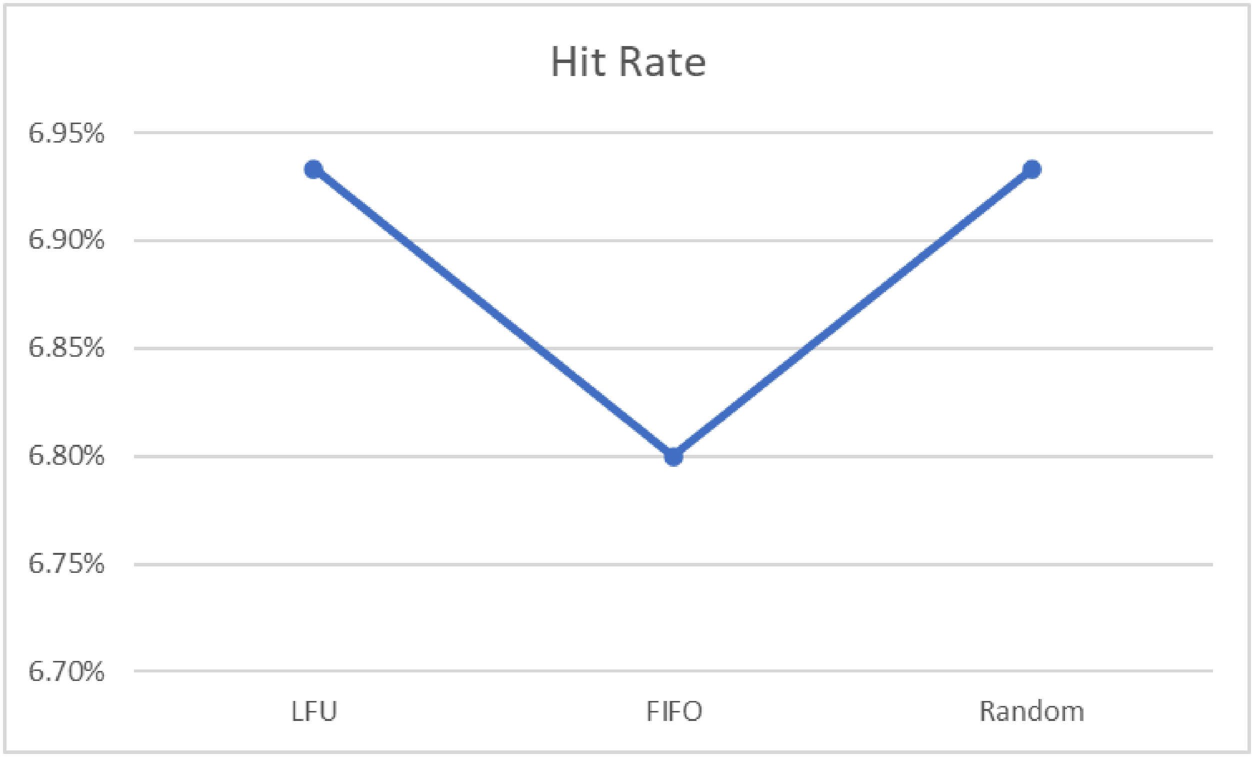
Above is a summary table of the measured metrics for each cache strategy, LFU, FIFO, and Random Replacement.

Figure 1. *Average Cache Retrieval Time for LFU, FIFO, and Random Replacement Caching Strategies.*



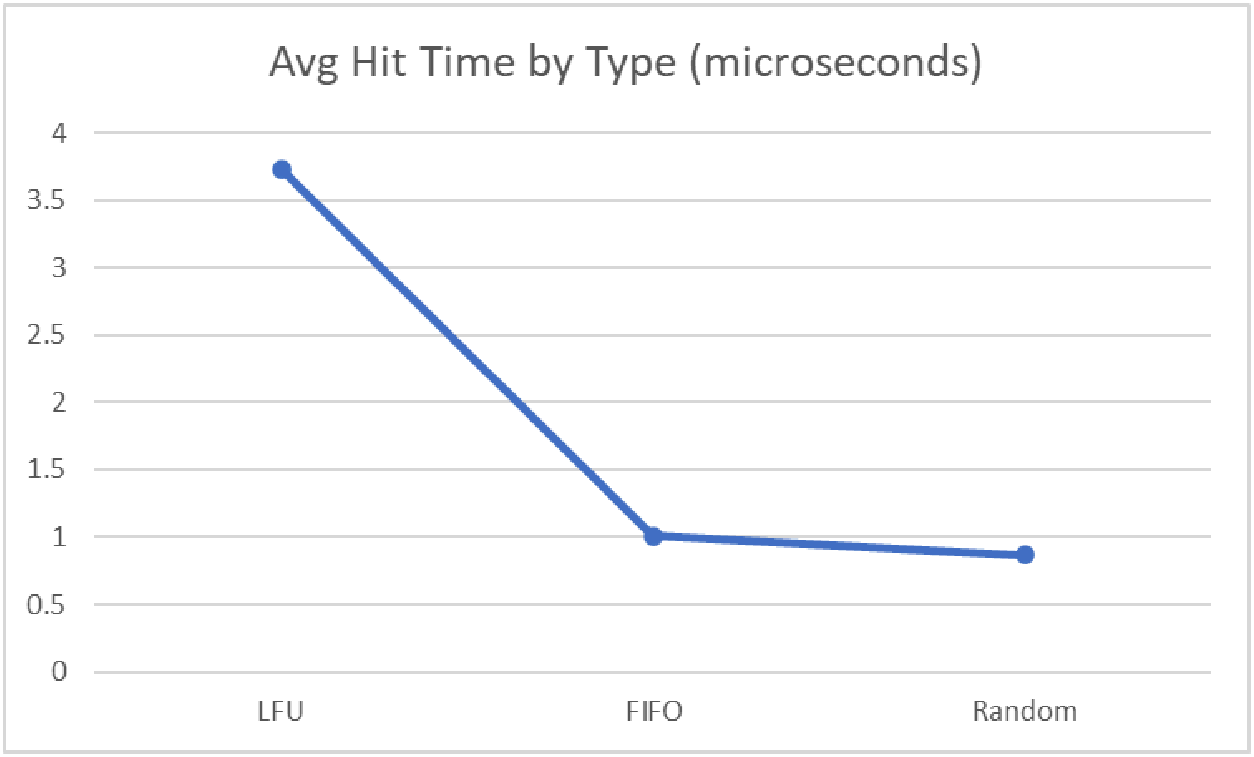
FIFO delivered the fastest overall average lookup time at 9.54 microseconds, Random Replacement was slightly slower at 10.39 microseconds, and LFU trailed at 14.75 microseconds. Note that the overall average is a weighted combination of fast hit times and slower miss times based on the hit rate, so it can be lower than the average miss latency when hits occur frequently enough to pull the average down relative to the miss-only timing.

Figure 2. *Cache Hit Rate for LFU, FIFO, and Random Replacement caching strategies.*



Despite differing eviction logic, all three strategies achieved similar hit rates on the randomized query set. LFU and Random both reached 6.93%, while FIFO followed closely at 6.80%. This demonstrates that policy choice had minimal effect on hit frequency.

Figure 3. *Average Cache Access Time (on hit) for LFU, FIFO, and Random Replacement caching strategies.*



Breaking down cache performance, Random Replacement had the fastest lookup time on hits at 0.86 microseconds, followed by FIFO at 1.01 microseconds, and LFU with the slowest at 3.73 microseconds. Figure 3 confirms that Random incurs minimal overhead on successful accesses, while FIFO still maintains efficient lookups. In contrast, LFU's hit performance is impacted by its frequency-based tracking.

Conclusion:

This experiment measured the efficiency of three cache replacement policies, LFU, FIFO, and Random, when serving population lookups for cities across 750 queries. All three policies were subjected to the same query set drawn from a shuffled and partially repeated dataset. With a small cache size of 10 entries and moderate hit rates (6.8–6.9%), the replacement strategy had a measurable effect on performance.

FIFO consistently delivered the best overall average lookup time due to its simple and predictable eviction pattern. Random performed nearly as well, especially excelling in hit latency. LFU, while designed to prioritize frequently accessed entries, incurred significant overhead due to frequency tracking and delivered the slowest average lookup time.

In practice, when hit rates are low and the working set is large relative to the cache size, simpler strategies like FIFO or Random may outperform more complex policies like LFU. This insight is particularly useful in systems where lookup speed is more critical than optimizing for hit frequency. Further testing with higher locality workloads and larger caches could help determine if LFU's complexity becomes worthwhile under different conditions.