CS 2510 - Project Proposal

Ziwei Quan, Shuhao Yu

Overview

In this project, we have chosen to reimplement a paper titled 'FIFO Queues are All You Need for Cache Eviction,' which was published at SOSP2023. The paper proposes a method for quickly evicting data accessed only once to improve the hit rate of FIFO. Caching is a crucial component in distributed systems, such as reducing communication overhead by caching hot data; solving data consistency issues with appropriate caching strategies to reduce the cost of data migration and synchronization. Our plan for the project is to understand this algorithm and compare its performance under different parameters (such as cache size) through experiments. Ultimately, we aim to gain a deeper understanding of cache eviction algorithms.

The project is planned to last four weeks: the first week involves team communication to choose the topic; the second week is dedicated to a detailed study of the paper and understanding the algorithm flow; the third week involves testing experiments; and the fourth week is for writing the report. Most of the work will be done collaboratively.

Paper Summary

The paper addresses the inefficiencies of existing cache eviction methods, such as LRU not being scalable and not being scan-resistant, and FIFO having a high miss ratio. The problem is to optimize the FIFO algorithm to swiftly evict data that has not been accessed after insertion, ensuring that the cache contains predominantly frequently accessed data to improve the hit ratio.

The paper's contribution lies in demonstrating that for cache workloads with skewed popularity, most objects are accessed only once before eviction. Therefore, quick demotion of such objects is crucial for cache efficiency.

Based on observations, the paper introduces the concept that shorter caches tend to have a higher "one-hit wonder" ratio. It proposes the addition of a shorter queue to determine if data inserted has not been accessed since insertion. Furthermore, to give such data a second chance, a "ghost" queue is maintained, which records hash keys without the actual data. When it's found that data in the short queue has not been accessed, it is moved to the ghost queue. When data is accessed and its hash key is still stored in the ghost queue, the data can be directly moved back to the main queue.

Project Proposal

In this project, we want to explore the performance and mechanism of cache in the distributed system. Meanwhile, we also want to test and recover the S3-FIFO on the local machine to practice the whole process of cache building and monitoring.

In the preparation step, there are two major open-source tools will be imported. First, cacheLib which is the pluggable cache engine to support the building and recording of S3-FIFO and other caches in the control group. Second, libCacheSim which is a similator to run different caches with various algorithms like LRU, FIFO, and TwoQ. And our experiment and test will be implemented

based on these tools.

There are three major metrics of a cache algorithm that we want to explore: byte miss ratio, throughput, and flash write friendliness. The miss ratio is the most important and straightforward factor of cache efficiency. It represents the ability and bandwidth usage of a cache algorithm. Throughput will show the highest request quantity of the cache. In other words, a higher throughput will reduce the pressure and workload on each core of the CPU. Flash write is one interesting and special test case. In this part, we plan to choose the dataset that contains enough flash write operations.

In addition, one of the significant structures of S3-FIFO is the ghost cache. During the test, we also plan to modify the length of the ghost cache to accomplish an inner comparison of S3-FIFO.

Groundwork

The guidelines of libCacheSim recommend users to run and implement the performance tests on the Linux kernel system. In this project, we plan to setup all cache algorithms and cache movement implements on Ubuntu 20.04 LTS which is a classical version for most similar testbeds. To accomplish the environment of Ubuntu, a system simulator VMware Workstation Pro will support the performance of the whole experiment.

And the designer of S3-FIFO also provides enough dataset resources to run their algorithm with libCacheSim. Thus, the traces that will be implemented in libCacheSim come from the Parallel Data Laboratory. Matplotlib will plot the visual and clear analysis and comparison for all results. We plan to use throughput, miss ratio, and flash-friendliness to analyze the performance of different algorithms.

The first potential risk in the project is the limitation of the testbed. Comparing with a one-million-core cloud lab, the stress test and size of the dataset may cause overload of the CPU and lack of the RAM to keep the test. Besides, the substitution of the testbed Ubuntu may not fully handle potential resources in the test.