# MRC: Parallel Cache Simulation for Miss-Ratio Curves

Parallel Functional Programming | Fall 2021 | Final Project Proposal

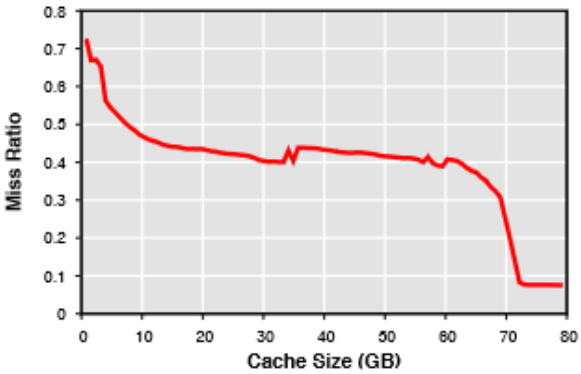
Jeffrey Tao
jat2164

Kaylee Treviño
kt2846

## Goals

In this project, we aim to perform cache simulations in parallel on a variety of workloads, varying cache eviction algorithm and cache size to produce a miss ratio curve per-algorithm per-workload.

## Background




Miss ratio may fluctuate in unexpected ways for a given cache algorithm and workload across cache sizes. Cache misses invariably translate to higher latency for a certain data-dependent operation. To this end, Miss-Ratio Curves (MRCs) are a useful tool for profiling how a given cache eviction policy performs on a given workload. The miss ratio is expressed as:

$$\frac{\#\ of\ misses}{\#\ of\ total\ operations}$$

A MRC can be produced by running the workload on a simulated cache. As the simulation runs, it simply tracks whether each cache operation is a hit or a miss, and computes the miss ratio after it finishes simulating the workload. This produces one point in the MRC. The full MRC is produced by doing this simulation at every cache size between a lower and upper bound, usually from 0 (100% miss rate) to the total size of the data touched by the workload (0% miss rate).

## Scope of Work

An attractive characteristic of this project plan is how extensible it is. Once the basic trace loading and cache simulation implementation is complete, we can always add or cut work by adding new cache eviction algorithms, new traces & adapters, and new simulation features such as sampling.

### Trace Adapters

Production cache access traces are available, but do not have a standardized format. Each cache needs a simple adapter to produce a unified format that our simulation function accepts. Each trace adapter will produce a

list: `[(id, size)]`, where `id` uniquely identifies an item in the workload used for lookups and evictions, and `size` is the size in bytes of the item.

## Eviction Algorithms

There are a variety of potential eviction algorithms that we might like to implement. Many algorithms require some state for bookkeeping. We can conceptualize the cache eviction algorithm as a pure function:

```
f(state, cache contents, next access) →
        (state, eviction choice)
```

State is opaque to the simulation runner and is algorithm-specific. As such, it is simply stored after an invocation of the eviction algorithm and passed back into the next invocation. An example of state is the priority queue for the Least Recently Used algorithm.

## Cache Simulation

The cache simulator is initialized with the cache size, eviction algorithm, and workload. It creates a representation of the (initially empty) cache contents and begins simulating the workload. For each access in the workload, it tracks if the access is a hit or miss. If it is a miss and the cache is full, it invokes the eviction algorithm and applies the eviction choice to the cache contents. The simulator continues this process until the workload trace is completely consumed. At the end, the simulator returns the miss ratio. Formally:

```
simulate(algorithm, size, workload) →
                    double
```

As final output, once all of the simulations across cache sizes for a given workload and eviction algorithm are complete, the program outputs all of the data points as a complete MRC. We can then visualize the data points separately as MRC plots.

# Parallelizability

Simulation of a particular eviction algorithm over a given workload trace is necessarily serial. However, the generation of MRCs requires repeating simulations at different cache sizes, which can be parallelized. In general, each individual simulation takes as input the intersection of three parameters selected from the sets:

- Eviction Algorithm (A)
- Cache Size (S)
- Workload Trace (W)

Hence, we have $|A| * |S| * |W|$ total simulations which can be performed in parallel.

# References


- Waldspurger, Carl A. et al. “Cache Modeling and Optimization using Miniature Simulations.” *USENIX Annual Technical Conference* (2017).
- twitter/cache-trace: A collection of Twitter's anonymized production cache traces.
- cache2k/cache2k-benchmark: Benchmarks for cache2k and third-party Java caching products
- sunnyszy/lrb: A C++11 simulator for a variety of CDN caching policies.