When is the Cache Warm? Manufacturing a Rule of Thumb

Lei Zhang∗, Juncheng Yang♭, Anna Blasiak‡, Mike McCall†, Ymir Vigfusson∗

∗Dept. of Computer Science, Emory University
♭Computer Science Department, Carnegie Mellon University
‡Indigo Ag †Akamai Inc ♯Facebook Inc

Abstract

The plethora of parameters and nuanced configuration options that govern complex, large-scale caching systems restrict their designers and operators. We analyze cache warmup times that can arise in failure handling, load balancing, and cache partitioning of large-scale distributed memory and storage systems. Through simulation on traces from production CDN and storage systems, we derive rules of thumb formulas for designers and operators to use when reasoning about caches.

1 Introduction

Today’s caching systems, including storage systems, distributed databases, and content-delivery networks (CDN), are large and abound with configuration options that can be opaque not only to the engineers operating these systems, but also to their designers [3, 4, 14, 25]. Two tendencies are to either ignore the complexity and set parameters from ignorance and experience, or to treat the system as such a complex black box that it requires another black box, such as machine learning models, to interpret the potential impact of changes. Between these extremes are simple and intuitive approximate models, or rules of thumb, that are invaluable in many engineering fields to create intuitive and sufficiently correct understanding of the system.

Here, we derive a rule of thumb expression for cache warmup times, specifically how long caches in storage systems and CDNs need to be warmed up before their performance is deemed to be stable. They are important in several contexts. In distributed storage systems or CDNs, operators may wish to discern how quickly after downtime or maintenance the server becomes useful again for serving content. They may wish to reason about how long to duplicate cache traffic to a new or recently restarted node before it can serve real clients at an acceptable hit rate. During recovery or reconfiguration of cache nodes, they may also wish to estimate how long the back-end storage servers must sustain additional load. In this manner, warmup time estimation allows CDN operators to compute the required redundancy and extra capacity to maintain a level of service in failure scenarios. In a shared memory or storage system, as another example, dynamic cache partitioning is often used to allocate storage resources to different processes or tenants. Here, when the partitioning controller decides to allocate more space to a tenant, it takes a period of time until the steady-state cache performance catches up. The controller needs to be cognizant of this delay to avoid instability whereby the partition keeps changing based on incomplete feedback gathered before steady-state has converged.

We first provide a concrete definition of cache warmup time, that is, a cache server has warmed up when its cache hit rate over time is and stays comparable (within ε error) to that of an identical cache service that processed the same workload but suffered no downtime. We then analyze dozens of traces across workloads collected from diverse systems, ranging from block accesses of virtual machines in storage systems to cache accesses of large CDN providers. We derive the following rule of thumb expression for operators to estimate warmup time of an LRU-style cache:

\[ \text{warmup-time}(s, \varepsilon) \propto s^{\rho_s} e^{-\rho_p \varepsilon}, \]

where \( s \) represents the cache size, and \( \varepsilon > 0 \) controls how closely hit rate should match that of a hypothetical cache server which was continuously running. Our experiments show that the \( \rho_s \) and \( \rho_p \) parameters concentrate at specific values for each type of workload. Our simulation results indicate that the formula provides an accurate expression for operators to estimate their cache server warmup time.

2 Motivation and Background

Distributed memory caches are the cornerstone of today’s content distribution networks (CDNs) and cloud storage systems for improving web service performance. A common architecture for a distributed cache is a collection of high-memory servers which is interposed between client nodes
Operational dynamics. Most research on caches assumes they operate in steady-state. Yet understanding the cache behavior under exceptional circumstances is often crucial.

Failure recovery. First, distributed caches can comprise a vast number of servers [17], where individual server failures are common. Accurate assessment of recovery time becomes increasingly important for operators to decide when servers are ready for serving clients without imposing significant load on the storage layer or end-user perceived latency. We assume that the cache memory on the server is empty (cold) after recovery because stale cache data can produce application-level inconsistencies, even with sophisticated application-specific cache invalidation pipelines [15].

Load balancing. Second, consistent hashing does not account for key popularity, so some cache servers can become heavily loaded relative to others [12]. Manual or automatic adjustment of hash ranges to balance load [10] implies that some cache servers are responsible for key-value pairs they have not encountered before, thus impacting cache hit rate.

Cache sharing. Third, large cache installations are often shared between multiple applications or tenants to improve efficiency and quality of service, either implicitly [1] or explicitly [5]. Explicit sharing is implemented via cache space partitioning mechanisms [8] which means cache space allocation for tenants may change over time. Operators must estimate how regularly cache space can be re-partitioned, which in turn depends on how quickly the enlarged cache space for tenants becomes useful and indicative of the tenant’s cache hit rate performance under steady-state [6].

Cache dynamics. Operators facing these scenarios would benefit from a rule of thumb to estimate when partially full cache memory has reached a “useful” steady-state and when applications can use the cache without burdening the storage layer or imposing miss latency on clients. Yet, quantifying cache warmup time is challenging due to several factors.

Cache hit rate performance is determined by the workload. Decades of effort has been spent on characterizing cache workloads, but historically focused on programmatic workloads (such as CPU caches) rather than in the context of human-driven behavior (such as web or CDN workloads) [11,20].

Cache workloads are not static. As mentioned earlier, considering a cache server to be warmed up when a particular

![Interval Hit Ratio (IHR) Curve](image)

**Figure 1:** Examples of Interval Hit Ratio (IHR) Curves. Each interval is 1/1000 of the original trace length.

fixed hit rate threshold is reached ignores temporal popularity dynamics [26] and diurnal variability exhibited in CDN traces [21], among others. Even defining hit rate relative to the start of a trace embodies the same problems.

Cache performance depends crucially on the cache size. Recent attention on efficiently computing so-called hit rate curves – hit rate as a function of cache space – has illuminated how the relationship tends to be nuanced and volatile in real-world workloads [7,19,27]. Bonfire [33] uses temporal and spatial behaviors for doing proactive cache warmup, but does not take cache size into account when defining warmup time.

3 Understanding Cache Warmup

Interval hit ratio. Warmup time must capture the notion of a cache “being useful”, which in turn is related to its hit rate. But the classical notion of “hit rate”, defined as the number of hits received over a number of accesses in a trace, relies on requests since the beginning of measurement being predictive of upcoming request—a degree of stability not present when cache workloads change dynamically.

To address variability in workloads, we measure cache performance by the interval hit ratio (IHR), defined as the ratio of cache hits in a relatively short past time window divided by the total number of requests in that window. This focus on the recent past adapts the metric to measure current performance with the ongoing dynamics. The IHR can be considered over subtraces of the full cache trace. The added flexibility allows us to also consider cache downtime, represented by a specific interval during the workload. We use $IHR(st,et,s)$ to denote the hit ratio computed for a short interval between start time $st$ and end time $et$ at cache size $s$. Below, each interval spans 1/1000 of the trace length.

Our analysis shows that even within the same workload type, workloads usually behave differently in terms of smoothness of the IHR curve. Figure 1 depicts the IHR curves of four workloads, two from Storage1 workloads and the other
two from CDN1 (see description in Section 4). We can see that the IHRs of CDN1-a and Storage1-a workloads remain high in most intervals, but CDN1-b and Storage1-b workloads are generally more fluctuated, even considering the periodic processes as a multi-day trace in the Storage1-b workload.

We note that in our analysis, we internally compute hit rate curves, or hit rate as a function of cache size, which can be efficiently generated through spatial sampling of the cache trace [27,32]. Spatial sampling could be used for online computation of the warmup time if needed.

**Cache warmup time.** We are now ready to define cache “warmup” time using the interval hit ratio. At a high-level, we declare a particular moment in the trace as when the server comes up (with an empty cache) after downtime. We then compare the IHR of that server (a downcache) to that of a server that did not go down at all (an upcache) while processing the exact same workload. When the IHRs of these two caches are sufficiently close, they have converged (as shown in Figure 2). This two-cache comparison overcomes the dynamical issues from above: a cache is consider warmed up if it behaves practically like one that did not go down.

Formally, we measure the difference in performance of a downcache that resumed operations at time \( st \) and an upcache by measuring the difference \( |IHR(st, t, s) - IHR(0, t, s)| \) at time \( t \). To capture the IHR of the upcache and downcache staying close, we define a tolerance parameter \( \varepsilon \) to express the maximum percentage difference we accept after warmup.

**Definition 1.** For cache size \( s \) and tolerance level \( \varepsilon > 0 \), a downcache that recovers at time \( st \) is considered warmed up at time \( t \) if for any end time \( et > t \), we have

\[
|IHR(0, et, s) - IHR(st, et, s)| < \varepsilon.
\]

Cache warmup time therefore depends on static factors, including cache size and tolerance levels, and dynamic factors dependent on the trace-based characteristics.

**Comparing cache warmup to fill up times.** The definition further highlights that downcache need not necessarily be filled for the cache to be considered warmed up: the rate of requests to items to which only the upcache was privy may simply be sufficiently limited that the downcache already contains the current working set and can be considered warm. An example is shown in Figure 3. Here we define cache is filled up when the cache capacity is fully occupied after a restart, whereas warmed up refers to the definition with \( \varepsilon = 1\% \). Across all our traces, the cache warms up faster than it fills up, with on average 39.1% and 36.8% for CDN1 and CDN2 workloads, and 16.6% and 23.8% for Storage1 and Storage2 workloads. These results underscore the opportunity for reconsidering cache warmup times in practice.

4 Deriving the Rule of Thumb

We analyze cache warmup time on several workloads to derive a useful estimation formula. Specifically, we look for a rule of thumb that embodies the following attributes.

1. **Simplicity.** Contain only a small number of parameters and as few as possible to capture the dependencies while being intuitive and practical to compute.
2. **Accuracy.** Closely approximate warmup time.
3. **Generality.** Yield insightful warmup time estimates for other, similar workloads.

4.1 Methodology

**Traces.** To derive and evaluate our rule of thumb formula, we analyze a variety of CDN and storage workloads. CDN1 includes 31 HTTP request traces from Akamai, within a single geographic region, comprised of 34 servers located in 23 logical data centers. The workload is a mixture of traffic across
a wide variety of different content types, including streaming media, file downloads, and typical web content such as HTML, images, JavaScript, and CSS. CDN2 are traces from the Wikipedia CDN servers [23]. Storage1 comprises 106 week-long hypervisor-observed disk access traces in production storage systems [27]. Storage2 consists of 32 file system traces released by MSR Cambridge [16].

**Implementation.** We implement the warmup analysis tool on top of Mimircache [31], an open source Python cache profiler that helps to calculate IHRs of traces, making the simulation process lightweight. We use an Intel Xeon CPU E5-2670 v3 2.30GHz system for our experiments.

**4.2 Step 1: Relaxing Dynamic Factors**

Our problem space is still large and unwieldy for operators to navigate in practice. Rather than capturing the full range of the workload’s dynamic characteristics, captured by the start time $s_{t}$ parameter, we simplify the definition to compute the maximum warmup time over all possible start times.

**Definition 2.** Given cache size $s$ and tolerance degree $\varepsilon$, the warmup time of a cache server, $\text{warmup-time}(s, \varepsilon)$, is the smallest $t$ such that for every start time $s_{t}$ and any $e < t$,

$$|IHR(0, e, t, s) - IHR(s_{t}, e, t, s)| < \varepsilon.$$ 

The above definition is the expression for which we will derive a rule of thumb. Simplifying is critical to minimize the number of parameters and create a practical formula.

**4.3 Step 2: Approximating Static Factors**

Armed with a compact definition, we can now analyze how cache size and tolerance degree affect cache warmup time on our traces. In search for a simple representation, we apply log-linear regression to model the relationship between warmup time, the cache size and the tolerance degree.

We observed that the cache warmup time has a piece-wise linear relationship to size until it reaches a plateau at larger sizes. There, the cache takes longer to warm up, but only until the working set of the trace is captured. The relationship between cache warmup time and tolerance is approximately log-linear; plotting warm-up time on a log-scale vs. tolerance on a linear scale produced a straight line. Larger tolerance degrees produce shorter warmup times as expected.

These observations suggest the following relationship, where $C$, $p_{c}$, and $p_{s}$ are free parameters:

$$\text{warmup-time}(s, \varepsilon) = C \cdot e^{-p_{c}\varepsilon} \cdot s^{p_{s}}.$$  

(1)

**4.4 Evaluation**

We now consider our proposed rule of thumb and how it measures up against our desired attributes.

1. **Simplicity.** The equation above says it all: there are only three free variables and one term.

2. **Accuracy.** To determine accuracy of the model fit we use a standard $R^{2}$ likelihood test. The $R^{2}$ distribution is shown in Figure 4. We consider 80% as $R^{2}$-threshold of a significance fit, so passing the test means the formula is accurate for use. As shown in the result, most of our CDN traces passed the $R^{2}$ likelihood test, together with a branch of storage traces. The accuracy is higher when considering both parameters together.

3. **Generality.** Because $C$ is a normalization parameter driven by time resolution, we investigate the ranges of parameters $p_{c}$ and $p_{s}$ as shown in Table 1. Note that here we only consider the traces that passed the test. These results meet our generality goal for the proposed method.

**Applying the rule.** A recent set of papers focused on offline optimal analysis of caches have shown that workload characteristics and object features are helpful for quantifying cache behaviors and further improving cache algorithms [2, 9, 32]. In line with those ideas, the warmup time of a workload can be estimated in a two-step process. First, we calculate the warmup times through an offline simulator on workloads, or a sampled workloads for efficiency. This step could be implemented through a simple API like:

```
offline-results = SIMULATE(workload, params)
```
Here the parameters $s$ and $\epsilon$ are varied in a wide range. We then apply regression over the offline results to optimize a simple model to express warmup time over these parameters, for instance using the following API:

\[
\text{warmup-time} = \text{ANALYZE}({\text{offline-results}}, \text{params})
\]

A key problem is how to make this process efficient. We have shown that cache warmup time can be successfully estimated with a lightweight method, and that simple regression can provide sufficiently accurate results. With a rule of thumb formula, operators and designers can estimate warmup time with only a few parameters. We note that warmup time is calculated for each workload and reflects the internal characteristics and behavior of that workload, so a system operator may only need to follow this process if the workload behavior changes drastically. Also, we found that workloads that share similar behaviors also yield similar parameters for their rule of thumb formulas. For instance, two CDN workloads for the same service are likely to share the rule of thumb parameters.

5 Extension: Enlarging a Cache

We have assumed thus far that a downcache starts off empty, which is reasonable in cases where a failure occurred since items may be stale, expired or awaiting invalidation. When a cache is resized, however, such as during cache partitioning, tenants retain existing content in the cache after it is enlarged. How would growing the capacity of a cache that already contains useful data affect the warmup time?

To define warmup time in the context of cache enlargement, we want to compare a “downcache” (to be resized) with a “upcache” (fully resized). We augment the interval hit ratio definition to $IHR(st, et, s, rt, m)$, where $rt$ expresses the time when the cache is to be resized, and $m \geq 1$ expresses a multiple of its current cache size $s$. Chronologically over a request stream, a cache of size $s$ begins at time $st$, its capacity is grown at time $rt$ to a new size $m \cdot s$ and ends at time $et$. Now:

**Definition 3.** Given cache size $s$, size multiple $m$, and tolerance degree $\epsilon$, assume a cache server is initially run at size $s$ and then resized at time $rt$ to $m \cdot s$. The resized cache server is considered to be warmed up at time $t$ if for every resize time $rt$ and all $et > t$,

\[
|IHR(0, et, m \cdot s) - IHR(0, et, s, rt, m)| < \epsilon.
\]

Here, the warmup time is driven primarily by the starting size, multiple, and tolerance level. Focusing on the first two parameters that relate directly to the cache size change, we fix tolerance level to 1% in the following experiments.

A primary difference between the recovery and resizing cases is that a cache that went down will be fully able to serve content after collecting all $s$ items, whereas $(m - 1) \cdot s$ items are missing in the resized cache. We therefore consider whether there is a log-linear relationship between warm-up-time in the resized context and $(m - 1) \cdot s$, and use the above methodology to obtain (for $C, p_r$ as free parameters):

\[
\text{resized-warmup-time}(s) = C \cdot ((m - 1) \cdot s)^{p_r}
\]

Our experiments considered $m \in \{2, 3, 4\}$ and varied $s$. The $R^2$ distribution (Figure 4) shows that most traces passed the $R^2$ likelihood tests using spatial sampling rate of 1% per trace. Our results also show that the $p_r$ exponent parameter still concentrates differently for each workload catalog (Table 1).

6 Related Work

Warmup is an important component of the systems or frameworks of several recent systems, yet many papers either define a warmup period arbitrarily, discard the first portion of a workload [18, 23, 28], or apply a warmup mechanism without quantifying or evaluating such methods [22, 24, 29, 30]. Zhang et al. [33] provided a cache warmup mechanism based on cache recency and is closest to our work. Their method does not consider workload dynamics. To the best of our knowledge, our work is the first to provide a practical method for estimating cache warmup time, and derive a simple expression for engineers and scientists to use.

7 Conclusion

There are many scenarios where operators of large distributed caches must implicitly or explicitly reason about warmup time of a cache server. Here, we derive a novel rule of thumb equation based on empirical results on a variety of real-world traces that demonstrates a power-law relationship between warmup time and cache size, coupled with an inverse exponential discount based on the desired tolerance level.

We build an offline simulator to fit free parameters of the formulas, which is shown to be concentrated within each workload category, to provide a useful expression for back-of-the-envelope calculations for the expected warmup time of cache servers without unduly impacting end-user clients or storage servers with miss penalties. We plan to release the code as an open-source Python package to aid the operators and designers of modern large-scale cache systems.

Acknowledgements

We are grateful to our shepherd, Deian Stefan, and the IMC 2018 and HotCloud 2020 reviewers for constructive feedback. We also thank Irfan Ahmad, Carl Waldspurger, and Avani Wildani for useful discussions. This work was supported by NSF CAREER Award #1553579.
Discussion Topics

Our paper sets up a framework to try to understand the process of warming up a cache. This is a question that, to the best of our knowledge, has surprisingly small literature. Understanding where and when the question arises in practice, and what approaches are used would be of interest.

Original cache state. As discussed above, distributed caches usually contain many storage servers or memory layers, and may encompass many concurrent tenants or classes of workloads. Among these scenarios, a tenant’s cache size can be increased or decreased. How do we reason about the original state of the adjusted cache partition? If more cache capacity is allocated during repartitioning, is it more practical to eagerly clear out the surplus entries (which may belong to a different application or tenant) or to keep those entries loaded? If a cache server is down for a while and restarted, as discussed elsewhere [13], is there a practice of keeping possibly stale information around? If some cache capacity is decreased, what content is removed? Is the best choice to remove the least-recently used items? Tracking detailed data placement choices is difficult since distributed caches are hard to trace, but might doing so reveal other factors that could impact the warmup process?

Cache dynamics. A key idea of this paper is to define Interval Hit Ratio (IHR) to represent dynamic cache workload properties and create opportunities for further analysis. However, as we discussed, distributed cache behaviors are typically workload related. For example, for a specific workload, restarting a cache server at different stages of the trace could result in different warmup times. We observed these patterns in our Storage workloads, where our accuracy is lower than of CDN workloads. Although we do simplify our warmup time definition to account less for abrupt dynamics and show the efficiency of the result, a richer analysis of cache dynamics could produce a deeper understanding of the warmup process and provide more precise expressions for system developers. This extension would likely be of most interest to the maintainers of specific large-scale distributed cache systems. We look forward to discussions from experienced operators of this kind about how warmup time analysis, and more generally rules of thumb, could be useful in practice.

References


[32] Lei Zhang, Reza Karimi, Irfan Ahmad, and Ymir Vigfusson. Optimal data placement for heterogeneous cache, memory, and storage systems. Proceedings of