A Study of Caching in Web Applications

Karansinh Thakor
University of Massachusetts Lowell
karansinh_thakor@student.uml.edu

ABSTRACT:

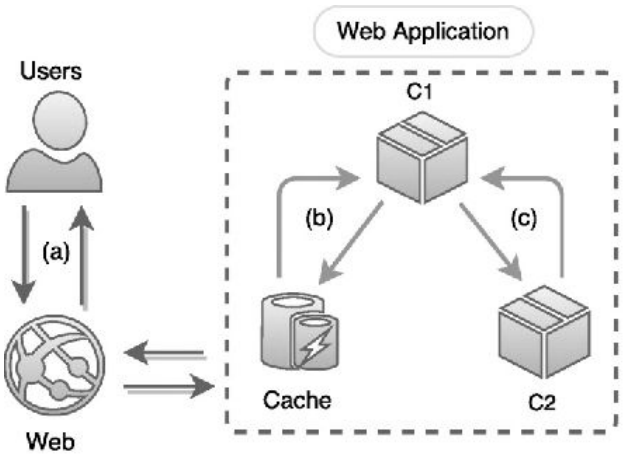
Nowadays a new form of caching is used called as application-level caching. It is used for improving application performance and stability. Caching is effective in improving performance because most applications are larger in size and perform repetitive operations. This operation includes IO activity which uses the larger portion of the CPU memory. The application includes temporarily storing base application code into the memory and decreasing web request. Leveraging caching require the knowledge of the domain and the application specificities. It is up to developers to manually manage cache with help of some external libraries. I have provided a quick study to the application and web caching. The goal is to not just use caching feature but also improving performance.

INTRODUCTION:

Due to a large number of users, the number of optimization techniques has been invented over the time. Dynamically generated content changes based on user request and due to the magnitude of the request, a performance of the server gets decreased. The solution to this problem is to use some form of caching. For enabling highly scaling web applications, frequently accessed data is cached into memory using a key-value. Caching is storing repeated operations in memory for later use, without the need for recurrent requests. It is the most common way improve application performance and scalability.

Many web caching solutions have been proposed. However, those solutions are used outside application as a transparent layer. Due to this fact, traditional caching is less efficient for complex web applications. Therefore with application-level caching is used to improve performance and stability. Application level caching solutions are developed in an ad-hoc way. The implementation of application level caching demanda high effort and it require manual implementation. Its design and maintenance involve some key challenging issues: where the cached data should be placed and maintained, what data should be cached, when the selected data should be cached or evicted and determining how to cache the selected data. There is no guide for developer to implement, maintain and design cachin in their application. There are only source like following traditional approach, development blogs and simply by searching online.

Given the issues involved with application-level caching, substantial advances have been made towards supporting developers while developing such caching solutions. In this article, I provide a comprehensive overview and comparison of existing approaches proposed in this context, so it is possible to understand what can be put into practice and remaining open issues. I first provide an introduction to web and application-level caching and then focus on surveying static and adaptive approaches in the literature. Thus, the scope of this survey has two key dimensions: static and adaptive application-level caching approaches. The former refers to non-adaptive solutions to help developers design, implement and maintain an application-level caching solution, typically focusing on raising the abstraction level of caching, with the provision of caching implementation support or automating some of the required tasks, thus providing caching management support. The latter is focused on cache management approaches that can adaptively deal with caching issues, usually by monitoring and automatically changing behavior to achieve desired objectives.



(a) Process Steps.

```
public class C1() {
  public Object process() {
  //creating the cache component
  Cache cache = Cache.getInstance();

  //looking up for cache content (steps in b)
  Object content = cache.get("c1:c2-computation");
  if (content == null){
    //cache miss (steps in c)
    content = C2.compute();

    //caching the content for future requests
    cache.set("c1:c2-computation", content);
  }

  //doing some business logic...
  return content;
  }
}
```

(b) Code Example.

INTRODUCTION TO APPLICATION-LEVEL CACHING:

As opposed to caching alternatives that are placed outside of the application boundaries, application-level caching allows storing content at a granularity that is possibly best suited to the application. For instance, modern web applications nowadays provide customized content and, in this case caching the final web pages is usually useless. We consider a web-based application any application that contains data and business logic maintained by developers or contains a presentation logic to provide content or features to users through the web. Therefore, application-level caching can be used to separate generic from specific content at a fine-grained level.

Application-level caching is mainly characterized by caching techniques employed along with the application code, that is, business, presentation, and data logic. Therefore, it is not tied up to a specific caching location (server-side, proxy, or client-side), because it can be conceived at the server-side, to speed up a Java-based application that produces HTML pages sent to users, as well as at the client-side, as a JavaScript-based application that executes part of its logic directly on the client's browser. In both situations, developers can reason about caching and implement a caching logic to satisfy their needs.

Thus, application-level caching has become a popular technique to reduce the workload on content providers, in addition to other caching layers surrounding the application. Such popularity is confirmed by Mertz and Nunes (2016), who analyzed 10 web applications and showed that application-level caching represents a significant portion of lines of code of investigated applications as well as a significant number of issues, considering that caching is essentially a non-functional requirement. In addition, Selakovic and Pradel (2016) analyzed 16 JavaScript open source projects and reported that 13% of all performance issues are caused by repeated execution of the same operations, which could be solved by means of application-level caching resulting in an improved performance and decreased user perceived latency.

Static vs. Adaptive Application-Level Caching:

A fundamental problem of application-level caching is that all issues mentioned earlier usually demand extensive knowledge of the application to be properly solved. Consequently, developers manually design and implement solutions for all those mentioned tasks. To provide solutions to deal with caching that require less intervention, and is thus easier and faster to be adopted, static

approaches have been proposed to help developers while designing, implementing, and maintaining an application-level caching solution. Static approaches usually focus on decoupling this non-functional requirement from the base code and providing ready-to-use caching components with basic functionalities and default configurations, automating some of the required tasks such as a storage mechanism and standard replacement policies. However, even when leveraging static solutions to ease the development of a caching solution, the issues and challenges related concerning design and maintenance remain unaddressed, because default configurations do not cover all the design issues, and those provided may not even perform well in all contexts.

Therefore, developers must still specify and tune cache configurations and strategies, taking into account application specificities. A common approach to develop a cache solution is to define configurations and strategies according to well-accepted characteristics of access and commonsense assumptions. Then, cache statistics, such as hit and miss ratios, can be observed and used to improve the initial configuration. This tuning process of caching configurations is repeated until a steady performance improvement is achieved. As a result, to achieve caching benefits so the application performance is improved, it is necessary to tune cache decisions constantly. Despite the effort to do so, eventually, an unpredicted or unobserved usage scenario may emerge. As the cache is not tuned for such situations, it would likely perform sub-optimally (Radhakrishnan 2004).

This shortcoming motivates the need for adaptive caching solutions, which could overcome these problems by adaptively adjusting caching decisions at runtime to maintain a required performance level. Moreover, an adaptive caching solution minimizes the challenges faced by developers, requiring less effort and providing a better experience with caching for them. Such adaptive caching solutions aim at optimizing the usage of the infrastructure, in particular for the caching system. Given that we introduced application-level caching and provided the background needed to understand the approaches that are discussed in this article, we now proceed to the presentation of different ways to provide a caching solution for developers as well as static and adaptive approaches that were proposed in the context of application-level caching.

CONCLUSION:

Application-level caching has been increasingly used in the development of web applications, in order to improve their response time given that they are becoming more complex and dealing with larger amounts of data over time. Caching has been used in different locations, such as proxy servers, often as seamless components. Application-level caching allows caching additional content taking into account application specificities not captured by off-the-shelf components. However, there is limited guidance to design, implement and manage application-level caching, which is often implemented in an ad-hoc way. In this paper, we presented a qualitative study performed to understand how developers approach application-level caching. The study consisted of the selection ten web applications and investigation of caching-related aspects, namely design, implementation and maintenance practices. We observed a higher number of categories, which are classes of observations made based on the analysis of these applications, associated with caching design and implementation than those associated with maintenance. Furthermore, the number of occurrences of each category, design and implementation categories also have higher numbers. This phenomenon was expected since the most representative portion of our qualitative data consists of source code and issues. Moreover, simple maintenance tasks and configurations are already executed and provided by external components, being commonly adopted. However, the number of occurrences do not reflect the importance of a category.