Seamless and Adaptive Application-level Caching

Jhonny Mertz and Ingrid Nunes (Orientador)

Programa de Pós-Graduação em Computação (PPGC)
Departamento de Informática Aplicada – Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{jmamertz,ingridnunes}@inf.ufrgs.br

Nível: Mestrado
Ano de ingresso no programa: 03/2015
Época prevista de conclusão: 02/2017
Data da aprovação da proposta de dissertação: Não aplicável
Eventos relacionados: SBES e SBCARS

Abstract. *Latency and cost of Internet-based services are encouraging the use of application-level caching to reduce the user-perceived latency and Internet traffic, and improve the scalability and availability of origin servers. However, this type of caching typically depends on specific details of the application and is designed and implemented in an ad-hoc way. Moreover, it is often tangled and spread all over the code. Although a well designed and implemented cache can achieve desired non-functional requirements, application performance may decay over time due to changes in the application domain, workload characteristics and access patterns, thus requiring constantly tuning the cache settings to cope with the changing dynamics of the domain, which implies extra time dedicated to maintenance. This work addresses these challenges imposed to developers by proposing a new framework to implement application-level caching, which aims to detach cache concerns from application code while leveraging application specificities to adapt cache to gradual changes at runtime.*

Keywords: *application-level caching, self-adaptive systems, separation of concerns, web application development*

1. Introduction

Dynamically generated web content represents a significant portion of web requests, and the rate at which dynamic documents are delivered is often orders of magnitudes slower than static documents [Ravi et al. 2009]. To continue satisfying users' demands and also reducing the workload on content providers, over the past years many optimization techniques have been proposed. The ubiquitous solution to this problem is some form of *caching* [Podlipnig and Böszörmenyi 2003]. Recently, latency and cost of Internet-based services are driving the proliferation of *application-level caching*, which can decrease the user perceived latency, and improve the scalability and availability of origin servers. Therefore, this type of caching has become a popular technique for enabling highly scalable web applications.

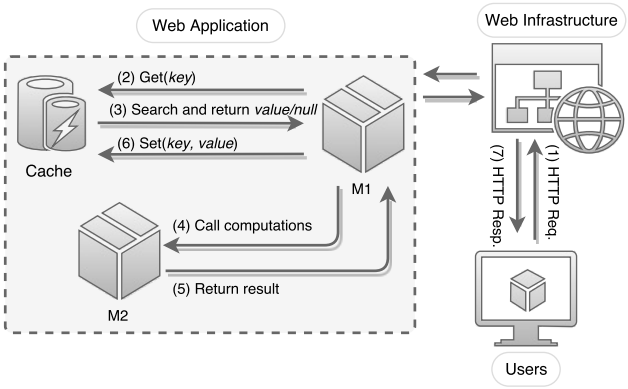
Despite its popularity, the implementation of application-level caching is not trivial and demands high effort, since it is typically implemented in an *ad hoc* way. It involves four key challenging issues: determining *what* data should be cached, *when* the data selected should be cached or evicted, *where* the cached data should be placed, and *how* to maintain the cache efficient. The main problem is that solutions for these issues usually depend on application-specific details and are manually designed and implemented by developers, which is time-consuming, error-prone and, consequently, a common source of bugs [Ports et al. 2010, Gupta et al. 2011]. Moreover, maintenance is compromised because caching logic is *tangled* with the business logic, and *spread* all over the code.

Although a well designed and implemented cache can achieve desired non-functional requirements, application performance may decay over time due to changes in the application domain, workload characteristics and access patterns. Thus, to leverage caching to improve application performance, it is required a frequent adjust of caching decisions, which implies extra time dedicated to maintenance [Radhakrishnan 2004]. This shortcoming motivates the need for adaptive caching solutions, which could automatically improve themselves to cope with the changing dynamics of the application while minimizing the challenges it poses for developers.

In Section 2 we give background on application-level caching and describe our problem. We describe research question and goals in Section 3, then present an overview of the proposed seamless and adaptive caching solution in Section 4. We describe the work in progress and outcomes expected from this study in Section 5. Finally, we discuss related work in Section 6 and, in Section 7, we conclude.

2. Background and Problem Statement

Figure 1(a) illustrates an example where an application-level cache is used to lower the application workload. First, the application receives an HTTP request (Step 1) from the web infrastructure. This HTTP request is eventually treated by an application component called M1, which in turn depends on M2. However, M2 can be an expensive operation (i.e. request the database, call a service or process that deals with a large amount of data). Therefore, M1 explicitly implements in M1's code a caching layer, which verifies whether the necessary processing is already in the cache before calling M2 (Step 2), as shown in Figure 1(b). Then, the cache component performs a look up for the requested data and returns either the cached result or a not found error (Step 3). If data is found, it means a cache *hit* and M1 can continue its execution without calling M2. If, however, a not found



(a) Process Steps.

```java
public class M1 {
 public Object process() {
   //instantiating the cache component
   Cache cache = Cache.getInstance();

   //looking up for cached content (steps 2 and 3)
   Object content = cache.get("m1:m2-computation");
   if (content == null) {
     //cache miss (steps 4 and 5)
     content = M2.compute();

     //caching content for future requests (step 6)
     cache.set("m1:m2-computation", content);
   }

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

(b) Code Example.

Figure 1. Application-level Caching Overview.

error is returned, it means a cache *miss* happened, then M2 needs to be invoked (Steps 4 and 5). The newly fetched result of M2 can be cached to serve future requests faster (Step 6) and eventually a response is sent to the user (Step 7).

As already discussed in the introduction, the development of application-level caching involves four key issues: determining *how*, *what*, *when*, and *where* to cache data. The first issue is related to the fact that the cache system and the underlying source of data are not aware of each other, as illustrated in Figure 1(a). The implementation and maintenance of application-level caching are thus a challenge because its direct management leads to a concern spread all over the system—mixed with business code—which leads to increasing complexity and time of maintenance, as presented in Figure 1(b). Furthermore, the extra logic also requires additional testing and debugging time, which can be expensive in some scenarios. Moreover, web applications are usually not conceived to use caching since the beginning. As the system grows, complexity or usage analysis are performed and may lead to requests for improvements. Thus, developers must refactor data access logic to encapsulate cache data into the proper application-level object. This on-demand and manual caching process can be very time-consuming, error-prone and, consequently, a common source of bugs [Wang et al. 2014].

The second and third issues refer to deciding the right content to cache and the best moment of caching or clearing the cache to avoid cache thrashing and stale content. These decisions involve (a) choosing the granularity of cache objects, (b) translating between raw data and cache objects, and (c) maintaining cache consistency, which might not be trivial in complex applications. The last issue is related to the maintenance of the cache system, which involves concerns such as determining replacement policies and cache size.

3. Research Question and Goals

Given the issues of application-level caching (hereafter referred to simply as "caching") mentioned in introduction, our research question consists of *how to seamlessly support developers in integrating caching concerns into the web applications and leverage application-specific information to achieve an adaptive cache management?*

This work addresses the challenges imposed on developers in the context of a new seamless and adaptive framework to detach caching concerns from the application

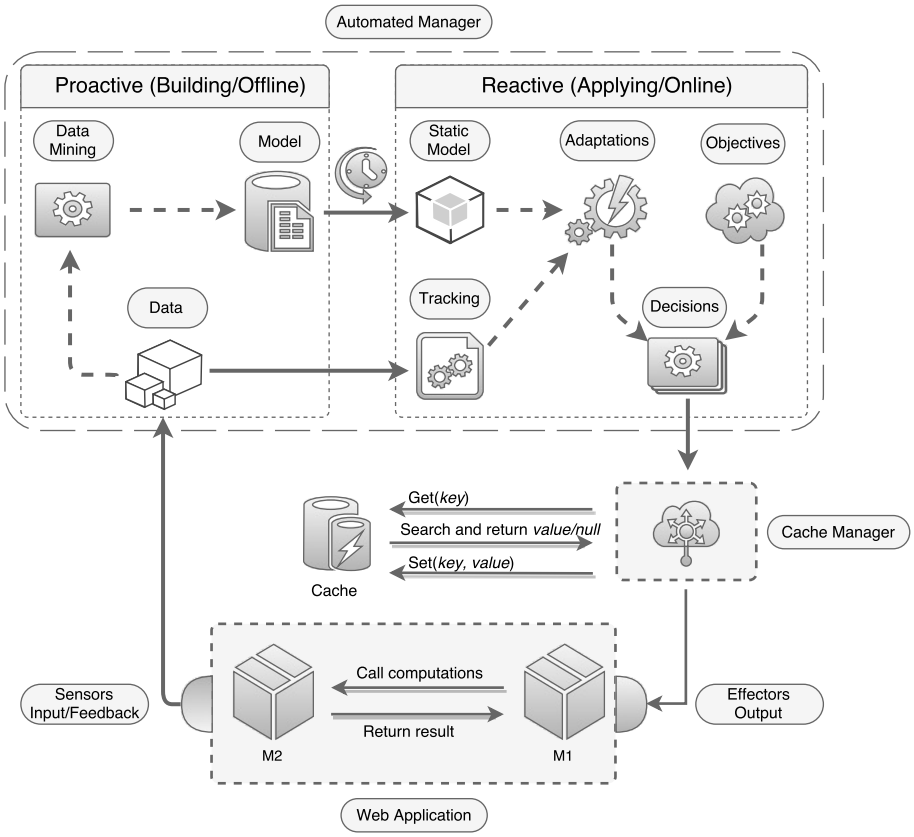


Figure 2. Seamless and Adaptive Application-level Caching Approach.

and adapt the cache management to gradual changes at runtime. Such adaptive solution would require minimal effort and input from developers, letting the caching abstractions decide the granularity of caching as opposed to providing a simple key-value store that developers must manually manage and tune when a supposed bottleneck is detected. In addition, it aims at an optimal utilization of the infrastructure, in particular of the caching system.

4. Proposed Approach

Our approach is composed of two key parts: a meta-model for obtaining application-specific information, and a runtime control loop to deal with caching concerns. The meta-model will capture declarative information about the cache and application, which will give hints to the autonomic system that in turn manages the cache at runtime autonomously. Figure 2 presents our approach, consisting of two complementary asynchronously running cycles: (a) a proactive (offline) model building, which monitors and analyzes the behavior of the application at runtime leveraging information from the meta-model instance, and (b) a reactive (online) model applying, responsible for executing adjustments accordingly to meet certain desirable objectives. Static or dynamic information about the underlying system is acquired through a seamless caching framework, which is being implemented using Aspect-oriented Programming (AOP) to provide separation of concerns.

To evaluate our proposed solution, we will perform a comparative analysis of

caching approaches: (a) manually designed and implemented by developers in existing open-source web applications (baseline), and (b) such applications with the proposed seamless and adaptive approach (without the originally implemented caching concerns). These caching setups will be evaluated regarding well-known cache metrics such as latency saving ratio and hit ratio, which will be measured by the execution of synthetic workloads, generated based on parameterized and empirical distributions.

5. Preliminary Results and Expected Contributions

An essential step towards such seamless and adaptive approach is to *understand*, *extract*, *structure* and *document* the application-level caching knowledge implicit and spread in existing web applications. Given this context, we performed a qualitative study, which involved the investigation of ten web applications (open-source and commercial) with different characteristics, to identify patterns and guidelines to support developers while designing, implementing and maintaining application-level caching. This qualitative research reveals that developers made use of supporting libraries and frameworks in order to prevent code tangling and raise the abstraction level of caching. Such supporting components are distributed cache systems and libraries that can act locally. Despite there are existing tool-supported approaches that can help developers to implement caching with minimal impact on the application code, cache decisions such as determining cacheable content and the right moment of caching or clearing the cache content are still designed and maintained manually, based on common sense, development blog consults, or online searches for tips.

These remaining issues motivate our adaptive application-level caching approach, which initially focuses on identifying and caching important methods (i.e. methods that would improve application performance if cached). The knowledge and metrics to distinguish important from less important methods were also derived from our qualitative study. In summary, the expected contributions of this work are: (a) a caching approach focused on integrating caching into web applications in a seamless and easy way, (b) a framework that detaches caching concerns from application code, and (c) an approach that provides an adaptive performance-driven selection and management of cacheable content by leveraging application-specific information captured in a meta-model.

6. Related Work

[Ports et al. 2010] and [Gupta et al. 2011] address implementation issues by providing high-level caching abstractions, in which developers can designate cacheable content, and the proposed system automatically caches and invalidates data. Aspect-oriented programming, which aims to increase modularity by allowing the separation of cross-cutting concerns, have been shown as an option to ease the caching implementation [Bouchenak et al. 2006]. Furthermore, popular web frameworks such as Spring Framework[1] and Rails[2] already provide solutions to help developers with caching implementation issues.

Although these approaches can raise the abstraction level of caching and prevent adding much cache-related code to the base code, their limitation is that they still require

[1] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/cache.html
[2] http://edgeguides.rubyonrails.org/caching_with_rails.html

reasoning about caching to decide whether to cache or not content. Our approach can minimize this problem by discovering cacheable spots, based on dynamic monitoring and analysis of the application and cache behaviors. [Della Toffola et al. 2015] addressed the identification of caching opportunities. Based on an analysis of profiling logs and predefined thresholds, the proposed approach identifies and suggests performance fixes (i.e. caching opportunities). However, developers should review the suggestions and refactor the code manually, inserting cache logic into the application.

These are static caching strategies, which may become less efficient in the light of unpredicted or unobserved workloads or access patterns. To supply these limitations, automatic and intelligent caching approaches have been proposed to dynamically configure strategies like admission and replacement policies [Podlipnig and Böszörmenyi 2003, Ali et al. 2011]. However, these proposed methods mainly address the context of web pages (at proxy level), providing good results in general but are less efficient where complex logic and personalized web content are processed within the application [Wang et al. 2014]. Nevertheless, these approaches can still inspire and provide insights to application-level caching adaptation.

Adaptive approaches towards application-level caching have been proposed to leverage replacement algorithms [Ma et al. 2014, Negrão et al. 2015], consistency management [Pohl 2005, Huang et al. 2010], and admission policies [Einziger and Friedman 2014]. Some approaches [Radhakrishnan 2004, Hu et al. 2015] focus on the cache infrastructure by exploring the size and fault-tolerance adaptations in cache servers. Even though these approaches focuses on providing adaptations at application-level caches, none of them takes into account application specificities to autonomously manage their target. Thus, they ignore cache meta-data expressed by application-specific characteristics, which are closely related to application computations. Consequently, caching design, implementation and maintenance shortcomings call for new methods and techniques, which can support developers to reach a good trade-off between cache development and application performance, leveraging application-specific details.

7. Conclusion

Our work focuses on supporting developers while designing, implementing and maintaining application-level caching in their applications. We expect to provide a better overall experience with caching for developers by automatically monitoring a web application and adapting the caching logic according to the changing dynamics. This work is the first step towards a context-aware approach that can manage the cache and application adaptively, minimizing the reasoning required from developers while reaching an optimal performance of the caching service on time.

References

Ali, W., Shamsuddin, S. M., and Ismail, A. S. (2011). A survey of Web caching and prefetching. *International Journal of Advances in Soft Computing and Its Applications*, 3(1):18–44.

Bouchenak, S., Cox, A., Dropsho, S., Mittal, S., and Zwaenepoel, W. (2006). Caching Dynamic Web Content: Designing and Analysing an Aspect-Oriented Solution. *Pro-*

ceedings of the ACM/IFIP/USENIX International Conference on Middleware, 4290:1–21.

Della Toffola, L., Pradel, M., and Gross, T. R. (2015). Performance problems you can fix: a dynamic analysis of memoization opportunities. In *Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 607–622, New York, New York, USA. ACM Press.

Einziger, G. and Friedman, R. (2014). TinyLFU : A Highly Efficient Cache Admission Policy. In *Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 146–153, Torino. IEEE.

Gupta, P., Zeldovich, N., and Madden, S. (2011). A trigger-based middleware cache for ORMs. In *Proceedings of the 12th ACM/IFIP/USENIX International Middleware Conference*, volume 7049 LNCS, pages 329–349, Lisbon, Portugal. Springer Berlin Heidelberg.

Hu, X., Wang, X., Li, Y., Zhou, L., Luo, Y., Ding, C., Jiang, S., Ding, C., and Wang, Z. (2015). LAMA : Optimized Locality-aware Memory Allocation for Key-value Cache. In *Proceedings of the USENIX Annual Technical Conference*, pages 57–69.

Huang, J., Liu, X., Zhao, Q., Ma, J., and Huang, G. (2010). A browser-based framework for data cache in web-delivered service composition. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2010*, pages 1–8. IEEE.

Ma, H., Liu, W., Wei, B., Shi, L., Bao, X., Wang, L., and Wang, B. (2014). Paap. In *Proceedings of the International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 983–986, New York, New York, USA. ACM Press.

Negrão, A. P., Roque, C., Ferreira, P., and Veiga, L. (2015). An adaptive semantics-aware replacement algorithm for web caching. *Journal of Internet Services and Applications*, 6(1):4.

Podlipnig, S. and Böszörmenyi, L. (2003). A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4):374–398.

Pohl, C. (2005). *Adaptive caching of distributed components*. PhD thesis, Dresden University of Technology.

Ports, D. R. K., Clements, A. T., Zhang, I., Madden, S., and Liskov, B. (2010). Transactional Consistency and Automatic Management in an Application Data Cache. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, pages 279–292, CA, USA. USENIX Association.

Radhakrishnan, G. (2004). Adaptive application caching. *Bell Labs Technical Journal*, 9(1):165–175.

Ravi, J., Yu, Z., and Shi, W. (2009). A survey on dynamic Web content generation and delivery techniques. *Journal of Network and Computer Applications*, 32(5):943–960.

Wang, W., Liu, Z., Jiang, Y., Yuan, X., and Wei, J. (2014). EasyCache: a transparent in-memory data caching approach for internetware. In *Proceedings of the 6th Asia-Pacific Symposium on Internetware on Internetware*, pages 35–44, New York, New York, USA. ACM Press.