Understanding and Automating Application-level Caching

Jhonny Mertz and Ingrid Nunes (Advisor)

[1]Programa de Pós-Graduação em Computação (PPGC), Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil

{jmamertz,ingridnunes}@inf.ufrgs.br

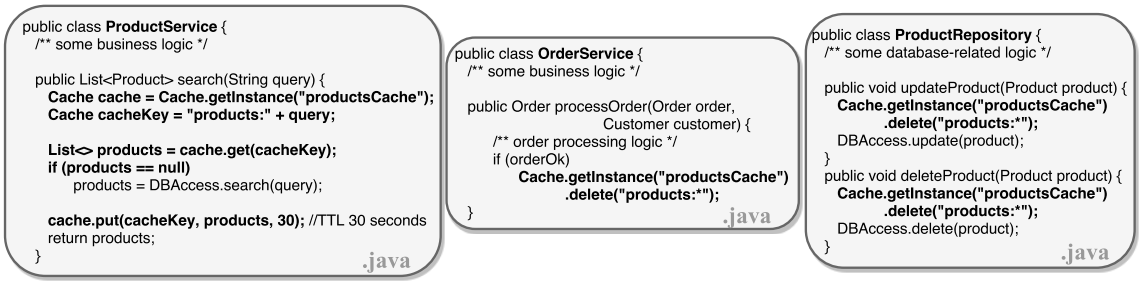
Abstract. *Application-level caching has increasingly been adopted to improve the performance and scalability of web applications. It consists of an additional caching layer that is manually added to the application code in selected locations. Because it requires a manual application analysis and selection of cacheable points as well as implementation, it is a time-consuming and error prone activity. In this paper, we introduce our key contributions in the context of application-level caching: (i) a comprehensive survey and taxonomy of work on this topic; (ii) a qualitative study that captures the state-of-practice of application-level caching, complemented by proposed guidelines and patterns; (iii) an adaptive component that autonomously manages admission of cache content; (iv) a framework that implements our proposal; and finally (v) an evaluation that provides evidence of the effectiveness of our proposal.*

1. Introduction

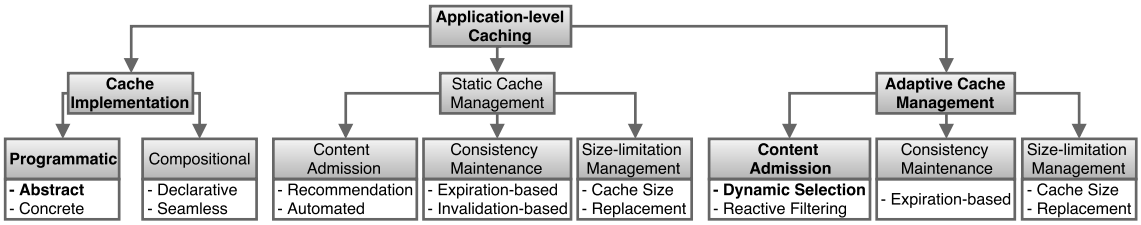
With the increasing popularity of web applications and distributed software systems, which deal with a high number of service requests, many techniques to improve their performance and scalability have been adopted. Such improvements are frequently obtained by means of caching. Recently, *application-level caching* has been explored as a new caching layer, expanding the granularity of cached content, such as methods, when compared to traditional forms of caching, being an application-tailored form of caching. In application-level caching, developers identify particular pieces of code that produce outputs that provide benefits if cached—because they are frequently accessed or expensive to compute—and manually manage a cache component and its content. Due to the required manual design and implementation, adding application-level caching to applications is not trivial, demanding high effort and expertise as well as deep knowledge of the application behavior. Not only are these activities manual but they are also done without proper guidance [Mertz and Nunes 2017a]. Moreover, because workload characteristics and request patterns change over time, implemented caching must evolve accordingly to keep providing improvements [Mertz and Nunes 2017b]. This calls for approaches that address the challenges of application-level caching and reduce the development costs.

In response, this paper outlines our research in this context. We provide guidance to adopt application-level caching with an in-depth survey and proposed patterns and guidelines. We also automate a key task of application-level caching, namely the selection of cacheable content. More specifically, we provide the following contributions:

- an understanding of state-of-the-art research on static and adaptive application-level caching with a proposed taxonomy of existing approaches;
- a qualitative study that captures the state-of-practice of application-level caching;



(a) Code Example



(b) Taxonomy of Application-level Caching Approaches

Figure 1. Application-level Caching Overview

- structured knowledge in the form of guidelines and patterns for the design, implementation and maintenance of application-level caching;
- an adaptive approach to automate the identification of caching opportunities;
- a framework built on top of up-to-date technologies, named APLCache, that allows developers to seamlessly integrate our approach with web applications; and
- an evaluation that gives empirical evidence of the effectiveness of our approach.

2. Taxonomy of Existing Work on Application-level Caching

To better understand the challenges associated with application-level caching, we introduce in Figure 1(a) an example in which this type of caching is used to lower the database workload. As can be seen, the caching logic is spread in different classes and tangled with the business code, causing maintainability problems. Moreover, developers must decide what to cache and identify when business operations require deletion of cached content.

To provide guidance to practitioners and promote research on caching at the application level, we performed a comprehensive survey of existing approaches [Mertz and Nunes 2017a], which were summarized, compared and categorized according to a proposed taxonomy (Figure 1(b)). Our taxonomy classifies approaches into three main groups. The first, *cache implementation*, refers to approaches that support the development of application-level caching by providing guidance or building blocks that ease its implementation, e.g. frameworks and libraries. The second and third groups include *static* and *adaptive* approaches, respectively, that (semi-)automate tasks of application-level caching. Adaptive (as opposed to static) cache management refers to approaches that evolve their output according to the dynamics of the application behavior.

Figure 1(b) highlights the categories to which our two proposed approaches—guidelines and patterns, and adaptive approach to cache content admission—belong. Existing work on the latter is limited to: (i) analyzing and caching methods of only specific types, such as those that access external application components; (ii) considering applica-

tions that have particular characteristics, e.g. are database-centric; or (iii) recommending caching opportunities (that need to be revised) by means of an off-line extensive profiling analysis. Our proposal, in contrast, does not make any assumptions regarding the methods to be cached or the target application and provides a fully automated caching at runtime.

3. Application-level Caching Guidelines and Patterns

Although existing cache implementation approaches provide ready-to-use components that reduce the effort needed to implement application-level caching, caching decisions as well as the maintenance of this type of caching are manually done in an *ad hoc* way. Nevertheless, many existing (large) software systems have incorporated application-level caching to improve performance and scalability. This practical knowledge can thus be used to evolve application-level caching development from an *ad hoc* to a systematic and disciplined way. We therefore performed a novel qualitative study to understand, extract, structure and document implicit application-level caching knowledge that is spread in existing applications. This allowed us to propose derived guidelines and patterns.

3.1. Foundations: Qualitative Study

Following the comparative and interactive principles of *grounded theory*, our study [Mertz and Nunes 2017a] involved the investigation of different types of artifacts from ten (open-source and commercial) web applications with different characteristics to take a holistic and comprehensive understanding of caching practices adopted by developers. We undertook mainly a subjective analysis of the data, collecting: (i) typical caching design, implementation and maintenance strategies; (ii) motivations, challenges and problems behind caching, and (iii) characteristics of caching decisions.

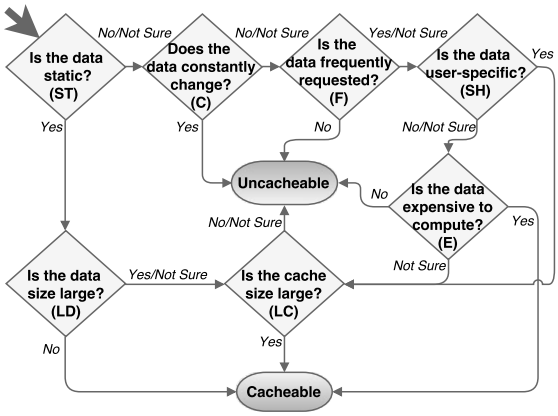
We investigated different application-level caching concerns, which are associated with the three key research questions: (1) what and when is data cached at the application level? (2) how is application-level caching implemented? and (3) which design choices were made to maintain the efficiency of the application-level cache? In addition to identifying recurrent styles of solutions (detailed as follows), there are interesting findings that reinforce the challenges associated with application-level caching, such as: (i) the indication, by developers, of uncertainty regarding what and when data should be cached, which can cause missed caching opportunities; (ii) the choice for simple cache design solutions (possibly non-optimal), possibly due to design effort and caching gains trade-off; and (iii) the presence of bugs and problems due to caching.

3.2. Catalog of Guidelines and Patterns

Our findings and observations were used as foundation for the provision of practical guidance for developers with respect to application-level caching [Mertz and Nunes 2017a]. In total, we derived 16 guidelines (high-level directions to develop caching solutions) and 4 patterns (systematic ways to address a caching issue) classified into three categories: design, implementation and maintenance. To illustrate, we show in Figure 2(a), the flowchart of one of the patterns, *Cacheability pattern*, which is used to help decide whether a method call should be cached.

4. Automating the Design and Implementation of Application-level Caching

Although the Cacheability pattern is a form of *abstract* reuse that helps to decide what to cache, questions associated with the pattern are not straightforward to answer as they



(a) Pattern Flowchart

Crit.	Evaluation (Informal Description)
ST	Yes, if the method always returns the same value when it receives a particular set of parameter values.
C	Yes, if the percentage of calls to the method returning different values for the same parameters is k standard deviations above the mean of this metric considering all methods.
F	Yes, if the number of stored traces of the method is above a required sample size, calculated based on a specified confidence level and margin of error.
S	Yes, if the percentage of calls to the method with the same set of parameter values is k standard deviations above the mean of this metric considering all methods.
E	Yes, if the mean time taken by a method to execute is k standard deviations above the mean of this metric considering all methods.
LD	Yes, if there is no enough space in the cache for the method returned value considering its estimated size.
LC	No, when LD is satisfied. LD and LC are evaluated together.

(b) Objective Criteria Evaluation

Figure 2. Cacheability Pattern

require a subjective evaluation of different criteria. Developers must thus be aware of and consider the application requirements, workload, domain, access patterns and business logic, which is still a challenge. To provide further support to cacheability decisions, we went beyond our pattern and proposed an approach that automates these decisions [Mertz and Nunes 2018], introduced next.

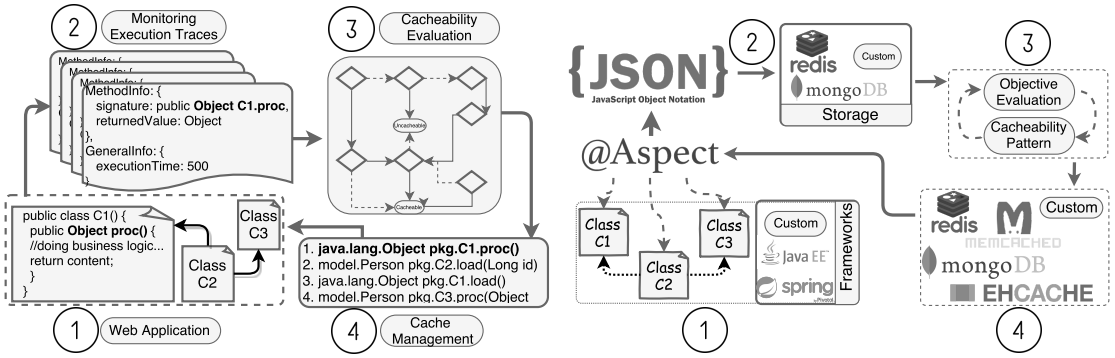
4.1. Adaptive Approach to Automate the Admission of Cache Content

Each decision in the flowchart presented in Figure 2(a) is associated with a criterion that must be evaluated to answer the decision question, they are: staticity (ST), changeability (C), frequency (F), shareability (SH), expensiveness (E), large data (LD), and large cache (LC). Our approach to automate the decision regarding the admission of cache content provides an *objective* means of evaluating each criterion. An informal description of how each of them is evaluated is summarized in Figure 2(b).

This objective evaluation of criteria is undertaken within a feedback loop of an adaptive component to be added to a web application. The loop has four main steps, illustrated in Figure 3(a): (1) the target web application is monitored at runtime; (2) traces are generated and stored for potentially cacheable method calls; (3) the cacheability of method calls is evaluated using our objective evaluation (Figure 2(b)) of the Cacheability pattern; and (4) the web application is modified to cache the selected method calls. Step 3 is executed periodically and, consequently, caching decisions evolve according to the current application usage patterns, thus making our approach adaptive.

4.2. APLCache Framework

Our approach, described above, was implemented as a framework, named APLCache, that can be instantiated to be seamlessly integrated with (existing) web applications. APLCache is implemented in Java and relies on a set of technologies that provide an appropriate infrastructure for the framework. The technologies used to implement each step of our approach are highlighted in Figure 3(b). To collect data to be analyzed and manage cacheable methods, our framework intercepts method executions using aspect-oriented programming. APLCache provides a set of alternative implementations based



(a) Conceptual Overview (b) APLCache Framework Technologies

Figure 3. Adaptive Approach to Automate the Admission of Cache Content

on the most popular web frameworks to obtain application-specific information. However, the implementation is decoupled from particular caching components, cache policies and algorithms, which can be configured through property files and annotations. Therefore, APLCache provides a fully customizable environment to be used with the proposed approach to automate the admission of cacheable content. The analysis process is performed asynchronously to prevent an application performance decay—it can even run on a dedicated machine.

4.3. Evaluation

We empirically evaluated our approach by simulating real-world workloads in three open-source web applications, from different domains and with varying sizes. Our simulations consisted of variations of simultaneous users constantly navigating through each application. We tracked *caching decisions*, measured the application *performance*, and compared the results obtained with APLCache and the cache manually implemented by developers (human-made decisions), using the application with no cache as a baseline. There is no automated approach to be compared with ours [Mertz and Nunes 2018].

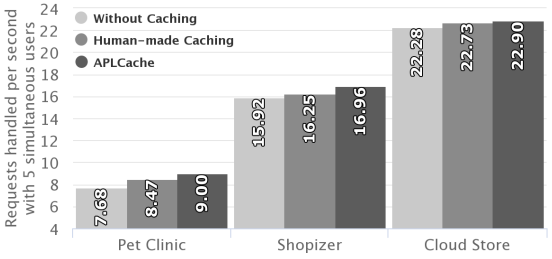
Figure 4(a) shows that our approach cached a higher number of cacheable methods (47%–300%) than the human-made decisions. It indicates that developers may be conservative while identifying cacheable methods, mainly because they cache all calls to the method, not just selected combinations of inputs and outputs as in our approach. Regarding performance, measured by throughput (number of requests handled per second) shown in Figure 4(b), our approach provided improvements in all cases, with gains ranging from 3% to 17% with respect to human-made caching. As conclusion, our approach not only reduced development time and effort, and thus costs, by automating a task that is currently manual, but also surpassed the improvements made by developers.

5. Final Remarks

Our work consists of substantial theoretical and practical advances towards the systematic development and automation of application-level caching, by providing contributions in many directions. First, it provides an introduction to this relevant topic with a survey and taxonomy of the state-of-the-art for both practitioners and researchers. Second, it investigates the state-of-practice leading to fruitful insights associated with application-level caching as well as structured knowledge (in the form of patterns and guidelines)

System	Total	H	A	H ∩ A
Cloud Store	812	4	8	4
Pet Clinic	205	1	4	1
Shopizer	5212	15	22	15

(a) Caching Decisions



(b) Throughput

Figure 4. Evaluation Results: Human-made Caching (H) vs. APLCache (A)

to support its development. Third, it proposes an adaptive approach that automates a key task associated with this type of caching, namely admission of cache content, which was implemented in a customizable framework and had its effectiveness demonstrated by empirical evaluation. The relevance of our work has been recognized by the software engineering community mainly through three publications in high impact international journals, according to the ISI Journal Citation Reports.

- ACM Computing Surveys [Mertz and Nunes 2017b] (*Qualis A1*), which has one of the highest impact factors among all computer science journals.
- IEEE Transactions on Software Engineering [Mertz and Nunes 2017a] (*Qualis A1*), which is considered the leading journal in software engineering.[1]
- Software: Practice and Experience (*Qualis A2*) [Mertz and Nunes 2018], an internationally respected journal with an emphasis on practical experience.

In addition to the great visibility of these journals, we presented our work at the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2017)—a top-level software engineering conference[1]—as a journal-first paper to further disseminate our work. Our initial ideas for this work were discussed at the Workshop of Thesis and Dissertations at CBSoft 2016 [Mertz and Nunes 2016]. Finally, our approach was used in a research project in the context of smart homes during a visit to the University of Grenoble [Mertz et al. 2017].

References

Mertz, J. and Nunes, I. (2016). Seamless and Adaptive Application-level Caching. In *VI Workshop de Teses e Dissertações do CBSoft (WTDSoft 2016)*, pages 70–76.

Mertz, J. and Nunes, I. (2017a). A Qualitative Study of Application-Level Caching. *IEEE Transactions on Software Engineering*, 43(9):798–816.

Mertz, J. and Nunes, I. (2017b). Understanding Application-level Caching in Web Applications: a Comprehensive Introduction and Survey of State-of-the-art Approaches. *ACM Computing Surveys (CSUR)*, 50(6):98:1–34.

Mertz, J. and Nunes, I. (2018). Automation of application-level caching in a seamless way. *Software: Practice and Experience*. https://doi.org/10.1002/spe.2571.

Mertz, J., Zapalowski, V., Lalanda, P., and Nunes, I. (2017). Autonomic management of context data based on application requirements. In *IECON 2017*, pages 8622–8627.

[1]See `doi.org/10.1016/j.infsof.2006.08.004` and `csrankings.org`.