Pinball Caching: Improving Performance of a Framework for Dynamic Web-Content Adaptation and Delivery

S. D'Antonio
ITEM Laboratory, CINI Consortium
Via Diocleziano 328
80124 Napoli, Italy
Email: sdantonio@napoli.consorzio-cini.it

M. Esposito
CRIAI Consortium
P.le E. Fermi 1
80055 Portici (NA) Italy
Email: m.esposito@criai.it

S. P. Romano
University of Napoli Federico II
Via Claudio 21
80125 Napoli, Italy
Email: spromano@unina.it

***Abstract*— In this work we deal with a multi-layer caching architecture capable to optimize delivery of dynamically produced web content. With reference to such architecture, we propose a technique named *Pinball Caching*, representing a useful means to estimate attainable performance improvement with respect to the trend of the main parameters characterizing the system. The application of such instrument allows clearly identifying where to concentrate efforts in order to optimize the joint utilization of the content adaptation architecture on one side and the hierarchical caching pool on the other.**

Keywords: Content Adaptation, Caching, Performance Evaluation.

I. Introduction

Networks and services are moving toward end-users, aiming at satisfying their needs with an ever-growing level of granularity and reaching more and more people as the time goes by. Ironically, the less the users technical skills the more the intelligence they expect from the network. For this reason, common people often have an irritating relationship with technology. From their point of view, a non-negligible delay perceived during a VoIP session is a big defeat, and they wonder why for years phone calls quality has been much better than nowadays. It is unlikely that a common person is able to place an order for something he likes, exploiting the current e-business infrastructures. And he would not ever understand the need for inserting login and password when buying milk.

Nevertheless, standing the above problems, it is more and more difficult to compete without the support of technology, regardless of the particular context. Technology is intrusive. It tends to pervade each field of human activities as witnessed by the proliferation of email and web addresses on buses, billboards, and business cards.

One of the great challenges of current IT research consists in easing anytime and anywhere access to technology by regular people in order to "move Internet from an opaque companion to a seamless, transparent partner in our daily lives" [1]. The main issue to face consists in leveraging current infrastructures to fill the gap between services and the concept users have about them, by exploiting an infrastructure which grants continuous, reliable and high quality access.

Research outlined in this paper is partially funded by the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) in the framework of the FIRB Project "Middleware for advanced services over large-scale, wired-wireless distributed systems (WEB-MINDS)"

A prerequisite underpinning the realization of the above envisaged scenario is the definition of an architecture able to provide optimal adaptation of contents in order for Internet services to be effectively accessed through the population of heterogeneous devices that are recently spreading. Such devices are progressively increasing Internet availability, but they demand for content adaptation in order to let services follow users, anywhere and at anytime, so as to foster pervasive computing scenarios.

In this work, starting from current achievements dealing with the definition of effective architectures for content adaptation and delivery (section II), we first introduce an extension for coping with highly dynamic contents (section III). Then, based on the consideration that high dynamicity implies heavy computation activities, we present a multi-layer caching approach, namely Pinball Caching (section IV), which helps mitigate performance figures. A heuristic analytical model is adopted in order to estimate the speed-up order of magnitude (section V). Finally, the paper presents a prototype implementation of the overall adaptation architecture (section VI), which also validates the pinball caching technique we introduced. In order to resume the achieved results, we provide in section VII some concluding remarks.

II. Related Work

Many existing works separately address content adaptation issues and content caching for performance improvement.

A complete architectural framework for content adaptation is presented in [2]. In order to assure client-side adaptation, this work defines a distributed architecture composed of different components which perform the necessary tasks along the service delivery chain. Our work draws some inspiration from this one. Though, it definitely departs from it in that the work in [2] suffers from the inability to effectively perform

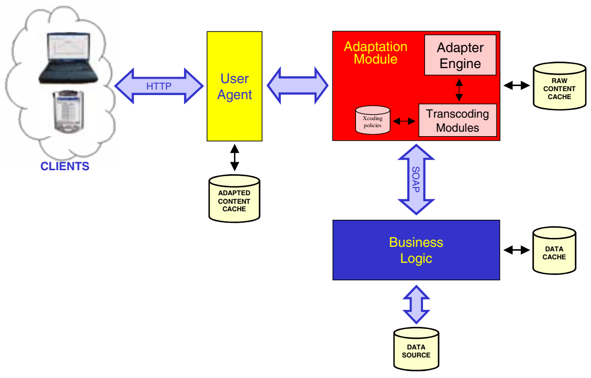


Fig. 1. The proposed content-adaptation and caching architecture

adaptation of fully dynamic contents. Only the possibility to deliver static contents, possibly available in several different copies for several different terminals, is natively considered by the cited architecture. We focus on a more dynamic context where content is possibly created on demand and, as such, it is highly tailored to the user's request. Furthermore, our content is supposed to be produced by a dynamic web-based application (e.g. query to search engines, web-mail, stock quotes surfing). In such a scenario, the availability of pre-existing available contents, obviously makes no sense.

In [3] the authors present a high level description of concerns in the context of web-content adaptation to mobile devices. The concepts of server-side, intermediate-side, and client-side adaptation are discussed and it is stated how a suitable mix of these three paradigms is mandatory in order to address a wide range of possible scenarios in the context of pervasive computing. This aspect fits our approach very well, since our presented architecture allows for a customizable distribution of adaptation tasks. Anyway, apart from some general considerations, in [3] the caching issue is not addressed at all. In our view, information management and delivery in such highly dynamic contexts impose the adoption of caching techniques, since caching is necessary in to preserve good performance. In [4], the authors present an approach to assure consistency in the context of dynamic caching activities. Their DUP (Data Update Propagation) algorithm, detects dependencies among objects and uses a tree-based structure to store them. Each object update triggers updates of dependent objects, which are pushed into the caches. This approach looks interesting to us, since we organize caches in multiple layers, and a cache update should trigger cascade updates in order to avoid high cache-miss rates, which might severely compromise overall system performance.

III. An Architecture for Dynamic Content Adaptation and Delivery

Our architecture is presented in Figure 1. It is composed of several modules which separate concerns along the service delivery chain. From the client perspective, the architecture allows to place web-requests to a User Agent. This module is in charge of managing a set of user-level facilities. Besides those belonging to any classical web-oriented system (e.g. authentication, authorization, accounting, user profiling), in this case the need arises for a thorough context-detection mechanism, including the following tasks:

- **device detection**, aimed at identifying hardware device capabilities and its current state (e.g. input/output device type, battery state);
- **context identification**, which detects current context variables (e.g. user-preferences, localization);
- **network channel identification**, describing the details about the available connections along the path between the User Agent and the user terminal.

The outcome of all these tasks influences the overall content-adaptation results, assuring the best possible matching between requested information and presented results.

The User Agent is transparent to the service semantics, and acts as a simple broker between users and the service providers (also opening an interesting market opportunity). In order to obtain the needed information, it forwards the user request together with the inferred context information to the service oriented-part of the architecture.

Upon reception of a request from the User Agent, the Adaptation Module extrapolates data related to context, and forwards the remaining part of the request to the Business Logic module. The request is herein managed, by accessing raw data from a generic data source. In this context, the data source is a generic source of raw information. Most likely, it is represented by a classical DBMS, but might also be an analytical model belonging to the targeted business realm. Once processed, the information coming back from the data source is passed to the Adaptation Module, which adapts it according to the previously stored context information. The result is sent back to the User Agent which simply forwards it to the user.

The operations performed inside each component of the architecture might be computationally intensive, depending on many factors: kind of requested information, raw-data processing algorithm, adaptation type, etc. This undermines scalability, i.e. the possibility for the architecture to handle a large population of different devices, which is a common case in a pervasive computing scenario. The presence of the User Agent module represents the first countermeasure we adopted to help mitigate this problem. In fact, User Agents are directly hosted by user domains and, due to their semantic transparency, they can be easily replicated and dimensioned according to the expected number of subscribers for each domain. In order to further improve the overall performance of the architecture, we decided to introduce a cache for each stage of the request fulfillment process. Caches are conceptually independent of each other and contain data at the level of abstraction of the stage they belong to. The cache level closer to the Business Logic module stores raw data, thus decreasing the total number of transactions executed on the data source (e.g. SQL queries to a DBMS). Most likely, this cache will contain the most interesting (e.g. the most recent, the most famous, or the most important) data. The middle-level cache stores data processed by the Business Logic. This cache will likely contain the most requested application-level information. For instance, it might contain the latest NASDAQ value and the trend of the five best stocks, even if these stem from processing the same raw data (containing the trend of all the stocks). Finally, the User Agent level cache contains the most used data presentations. This relieves the Adaptation Module from performing many times the same content-adaptation task. The fewer the classes of devices simultaneously active, the better the attainable performance improvement.

IV. The Pinball Caching Technique

In this section we analyse how caching helps improve the performance of the presented architecture.

Requests issued by users follow different paths depending on whether and where a request happens to raise a cache miss. In order to formalize all the possible cases, we defined the diagram depicted in Figure 2. Each request follows just one path through this diagram, from the topmost box ("Request") to the bottommost one ("End"), i.e. from user request until its fulfillment. This behavior resembles that of a ball in a pinball: after having shot the ball, it will eventually fall in the hole, but no one can predict through which of all the possible paths. For this reason we call this diagram the *Pinball Caching Chart*.

The first decision point ("Cacheable?") separates requests for cacheable content from those for non-cacheable one. For instance, if a request asks for a non-cacheable content (path 1 — e.g. a frame of a live video-surveillance session), the information necessary to compute the content is fetched from the data source; then, the raw content is produced, adapted, and sent back to the user. Alternatively, in case of cacheable content, the architecture verifies whether or not it is cached in the Adapted Content Cache. If it is (path 2), the answer can be immediately returned to the user (this is the case of a request for the same content and from the same type of device); otherwise, the content is searched for in the Raw Content Cache. In case of cache hit (path 3), the content needs just to be adapted, before being sent to the user (this is the case of a new device asking for an already processed content); otherwise, the content is eventually searched for in the Data Cache. In this case, a cache miss is dealt with in all respects as if the content were not cacheable (path 5).

We remark that in the pinball caching chart we placed two different boxes, related, respectively, to actions 2 and 5. This happens even though the actions themselves have the same semantics, i.e. transferring the adapted content across the network which connects to the client. For the sake of generality we leave the boxes separated because we take into account the case where the adapted content is cached in a separate network node, which might be far away from the user agent. In an even more general case, such adapted content might be stored in several caching nodes in case an approach à la Content Delivery Network is embraced. This stated, in the following of the paper we nonetheless consider the two above actions as indistinguishable (i.e. $t_2 \triangleq t_5$), which means that we are

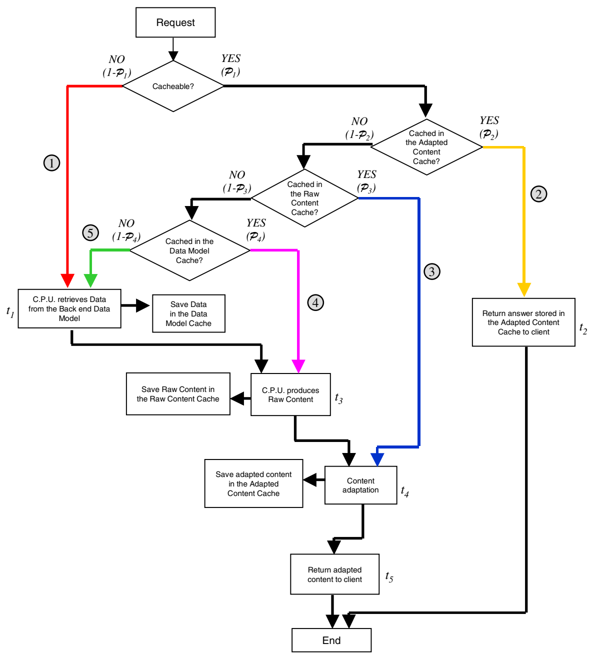

Request
NO (1-$\mathcal{P}_1$)
YES ($\mathcal{P}_1$)
Cacheable?
NO (1-$\mathcal{P}_2$)
YES ($\mathcal{P}_2$)
Cached in the Adapted Content Cache?
NO (1-$\mathcal{P}_3$)
YES ($\mathcal{P}_3$)
Cached in the Raw Content Cache?
NO (1-$\mathcal{P}_4$)
YES ($\mathcal{P}_4$)
Cached in the Data Model Cache?
1
2
3
4
5
t_1
C.P.U. retrieves Data from the Back end Data Model
Save Data in the Data Model Cache
Return answer stored in the Adapted Content Cache to client
t_2
Save Raw Content in the Raw Content Cache
C.P.U. produces Raw Content
t_3
Save adapted content in the Adapted Content Cache
Content adaptation
t_4
Return adapted content to client
t_5
End

Fig. 2. The Pinball Caching Chart

envisaging a scenario where there is just a single cache co-located with the user agent.

The overall response time figure depends on two groups of parameters:

- probabilities related to the decision blocks ($\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$, and $\mathcal{P}_4$);
- time spent to execute the various actions (t_1, t_2, t_3, t_4, and t_5); only non-negligible times are herein considered.

By appropriately defining such parameters, it is possible to evaluate how caching influences the overall performance.

Standing the presented diagram, Table I contains the expressions for t_{path_i} and $\mathcal{P}_{path_i}$ ($i = 1 \dots 5$), representing, respectively, the elapsed time and the probability to cross the i-th path.

i	t_{path_i}	$\mathcal{P}_{path_i}$
1	$t_1 + t_3 + t_4 + t_5$	$1 - \mathcal{P}_1$
2	t_2	$\mathcal{P}_1\mathcal{P}_2$
3	$t_4 + t_5$	$\mathcal{P}_1(1 - \mathcal{P}_2)\mathcal{P}_3$
4	$t_3 + t_4 + t_5$	$\mathcal{P}_1(1 - \mathcal{P}_2)(1 - \mathcal{P}_3)\mathcal{P}_4$
5	$t_1 + t_3 + t_4 + t_5$	$\mathcal{P}_1(1 - \mathcal{P}_2)(1 - \mathcal{P}_3)(1 - \mathcal{P}_4)$

TABLE I
ELAPSED TIME AND PROBABILITIES TO CROSS EACH PATH.

V. Qualitative Performance Evaluation

In this section we will carry out an analysis of the attainable performance improvement in the presence of our multi-layer caching architecture. We will take into account three different scenarios dealing, respectively, with the most generic envisaged situation, as well as particularly favorable and unfavorable scenarios.

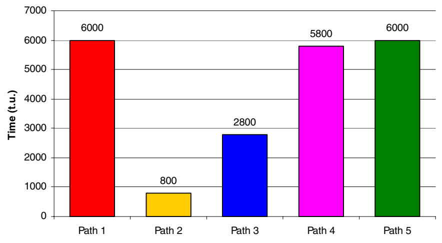


Fig. 3. Time elapsed across each path

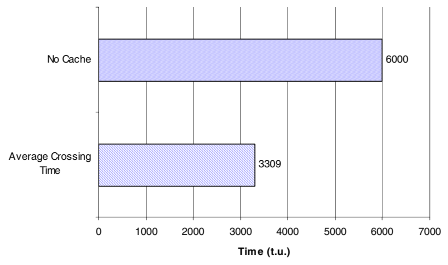


Fig. 4. Average time with and without caching in the generic scenario

A. Generic scenario

For our generic scenario, let us assume that the involved model parameters take the values in Table II.

Probabilities	Value
$\mathcal{P}_1$	75%
$\mathcal{P}_2$	50%
$\mathcal{P}_3$	60%
$\mathcal{P}_4$	70%
Actions	**Time (t.u.)**
t_1	200
t_2	800
t_3	3000
t_4	2000
t_5	800

TABLE II
CONFIGURATION OF PARAMETERS IN THE GENERIC SCENARIO

The values in the table have been chosen on the basis of the following considerations. Firstly, $\mathcal{P}_1$ (*Cacheable?*) has been set by considering that the probability of a cache miss is relatively small when accessing common web-based Internet applications; indeed, 25% cache misses look like a pessimistic estimate, thus representing an upper bound for our evaluation in a generic scenario. As far as probability $\mathcal{P}_4$ (*Cached in the Data Model Cache?*), we remark that such parameter just depends on the information acting as a source for building the requested content. That is to say, this parameter does not show any dependency on the specific processing that has to be carried out on the information building blocks. On the contrary, such a dependency plays a major role in the definition of probability $\mathcal{P}_3$ (*Cached in the Raw Content Cache?*), which is actually linked to the processing of such blocks. Finally, $\mathcal{P}_2$ (*Cached in the Adapted Content Cache?*) can be configured by taking into account both user's device capabilities and current user settings. In the light of the above considerations, it comes out that probabilities progressively decrease when climbing up the pinball caching chart, from the raw data level, towards the last stage of the content adaptation process.

With such configuration of parameters, we attain the results shown in Figure 3.

By looking at the chart, we can derive the following insights. Path 1 and path 5 present the same performance figures, i.e. a traversal time of 6000 time units. This is due to the fact that such paths, though for different reasons, refer to the absence of caching in the process of accessing the desired content: the former is crossed in case of a request for intrinsically non-cacheable content; the latter deals with the case of cacheable content not present in any level of the cache hierarchy. Path 4, referring to a hit in the data model cache, shows only a negligible improvement when compared to the previous cases. The reason behind such a behavior resides in the small time we set to access raw data at the data model layer (i.e. the time to perform a DBMS query). Path 3 just requests for adapting the content and sending it to the client (2800 time units). Best performance is achieved by path 2 since the already available content is just sent over the network (800 time units). The bottom part of the figure shows the traversal probabilities related to each path. By weighting the path traversal times with such probabilities, we derive the average crossing time. In Figure 4 it is compared with the time needed to fulfill requests in the absence of caching mechanisms.

As it comes out from the picture, the presence of caching, in the envisaged scenario, almost doubles the attainable performance.

B. Favorable scenario

For this scenario, we assume that the involved probabilities take the values in Table III. No change in the action times has been considered with respect to the generic scenario.

Probabilities	Values
$\mathcal{P}_1$	95%
$\mathcal{P}_2$	70%
$\mathcal{P}_3$	80%
$\mathcal{P}_4$	90%

TABLE III
CONFIGURATION OF PARAMETERS IN THE FAVORABLE SCENARIO

As opposed to the previous case, here we make the assumption that content has a high degree of reusability, which caters both for a very low probability for content not to be cacheable and a high cache hit probability. In this case, we have $\mathcal{P}_{path_1} = 5\%$, $\mathcal{P}_{path_2} = 66.5\%$, $\mathcal{P}_{path_3} = 22.8\%$, $\mathcal{P}_{path_4} = 5.13\%$, $\mathcal{P}_{path_5} = 0.57\%$, and the average times (shown in Figure 5), indicate an improvement of 3.33 times in the presence of caching.

C. Unfavorable scenario

For this scenario, we assume that the involved probabilities take the values in Table IV. Also in this case, no change in the

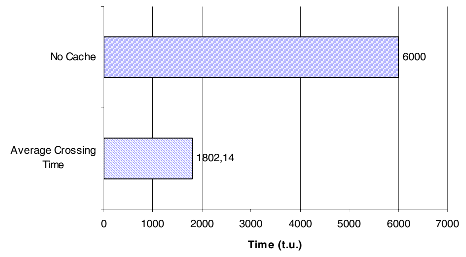


Fig. 5. Average time with and without caching in the favorable scenario

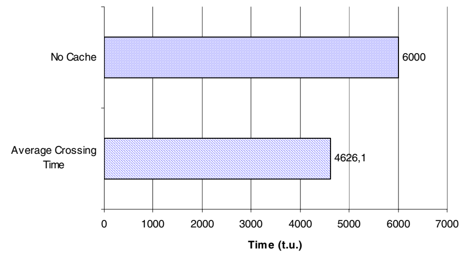


Fig. 6. Average time with and without caching in the unfavorable scenario

	Expression	Gen.	Fav.	Unfav.
$\frac{d\bar{t}}{dt_1}$	$(1-\mathcal{P}_1)+\mathcal{P}_1(1-\mathcal{P}_2)\cdot(1-\mathcal{P}_3)(1-\mathcal{P}_4)$	0.295	0.056	0.57
$\frac{d\bar{t}}{dt_2}$	$\mathcal{P}_1\mathcal{P}_2$	0.375	0.67	0.16
$\frac{d\bar{t}}{dt_3}$	$(1-\mathcal{P}_1)+\mathcal{P}_1(1-\mathcal{P}_2)\cdot(1-\mathcal{P}_3)$	0.4	0.11	0.68
$\frac{d\bar{t}}{dt_4}$	$(1-\mathcal{P}_1)+\mathcal{P}_1(1-\mathcal{P}_2)$	0.625	0.33	0.83
$\frac{d\bar{t}}{dt_5}$	$(1-\mathcal{P}_1)+\mathcal{P}_1(1-\mathcal{P}_2)$	0.625	0.33	0.83

TABLE V
COMPUTATION TIMES OPTIMIZATION

	Gen.	Fav.	Unfav.
$\frac{d\bar{t}}{d\mathcal{P}_1}$	-3588	-4419	-2498
$\frac{d\bar{t}}{d\mathcal{P}_2}$	-2418	-2474	-2123
$\frac{d\bar{t}}{d\mathcal{P}_3}$	-1147	-861	-1193
$\frac{d\bar{t}}{d\mathcal{P}_4}$	-30	-11	-46

TABLE VI
CACHE HIT RATIOS OPTIMIZATION

action times has been considered with respect to the generic scenario.

Probabilities	Values
$\mathcal{P}_1$	55%
$\mathcal{P}_2$	30%
$\mathcal{P}_3$	40%
$\mathcal{P}_4$	50%

TABLE IV
CONFIGURATION OF PARAMETERS IN THE UNFAVORABLE SCENARIO

We herein assume that content is highly dynamic, at the detriment of reusability, which translates in a very low cache hit probability. In this case, we have $\mathcal{P}_{path_1} = 45\%$, $\mathcal{P}_{path_2} = 16.5\%$, $\mathcal{P}_{path_3} = 15.4\%$, $\mathcal{P}_{path_4} = 11.55\%$, $\mathcal{P}_{path_5} = 11.55\%$, and the average times (shown in Figure 6), indicate a performance improvement of just 1.3 times in the presence of caching.

D. Sensitivity analysis of the model parameters

When approaching a system tuning phase, it can be interesting to evaluate where to concentrate efforts in order to improve performance. This kind of analysis will be carried out for the three scenarios presented. Useful insights will be derived from two different optimization strategies by focusing, respectively, on computation times and cache hit ratios.

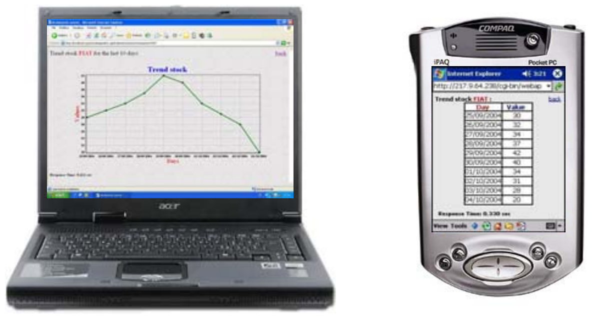
1) Computation times optimization: In Table V we report expressions and values for all the derivatives of the average response time with respect to each of the action times. The table suggests to focus the first stage of optimization on improving both the content adaptation process and the throughput of the transmission between the User Agent and the End-User Terminal. In fact, in both the general and the unfavorable scenarios, times associated with such operations (t_4 and t_5) have the strongest impact on the average response time. This comes out from the high probability that the related two actions are involved in the content delivery chain for a generic user request. On the other hand, in the favorable scenario, t_2 is the dominating parameter, since path 2 definitely has a higher probability than the other paths to be crossed. This emphasizes the importance of embracing a Content Delivery based approach in such case.

2) Cache hit ratios optimization: By using the same approach as in the previous section, we present in Table VI the values for all the derivatives of the average response time with respect to each of the probabilities (for the sake of conciseness, the derivatives expressions have been omitted in the table).

Since all probabilities refer to cache hit events, the table contains all negative values. In other words, an increase in hit probability at any of the cache hierarchy layers, always results in lower response times. Interestingly enough, the higher in the hierarchy the cache hit probability increase locates, the stronger the relative performance improvement achieved (e.g. increasing $\mathcal{P}_1$ has a stronger effect on the overall average response time than increasing any of the lower layer probabilities). As a final remark, we notice that optimizing caching mechanisms located at the lowest level of the hierarchy proves to be almost useless from the performance improvement perspective. The reason behind such consideration resides in the negligible impact of DBMS query execution on the overall response time.

VI. IMPLEMENTATION OF A CASE STUDY

In this section we present a proof-of-concept implementation of the hierarchical caching framework operating in the context of a web application capable to perform content adaptation. Our prototype is almost fully compliant with the layered caching architecture presented in Figure 1: for the



Trend stock FIAT :

Day	Value
25/09/2004	30
26/09/2004	32
27/09/2004	34
28/09/2004	37
29/09/2004	42
30/09/2004	40
01/10/2004	34
02/10/2004	31
03/10/2004	28
04/10/2004	20

Response Time: 0.330 sec

Fig. 7. Content adaptation in the presence of heterogeneous devices

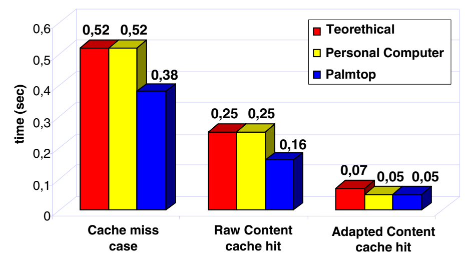


Fig. 8. Effect of hierarchical caching on response times

sake of simplicity it just avoids implementing the lowest-layer caching mechanism, i.e. the data cache (which however has no major influence on the overall achievements).

We take the example of a dynamic web site guaranteeing access to real-time information about stock quotes and their trends. The application on the server side is capable to automatically identify the client and serve its request accordingly. Interface between the user agent and the adaptation module, as well as all interfaces between server side components (adaptation module, business logic and data model) have been specified and implemented following the web services paradigm, i.e. by using protocols based on SOAP/XML [5].

Depending on the specific client capabilities, the retrieved raw content (i.e. stock trend information) is first adapted and then presented in the most suitable format. As an example, in case of a request coming from a user equipped with a PC or a notebook (see Figure 7), stock trend is presented in a graphical format; should the same request come from a device like a mobile phone or a palmtop, a tabular format is chosen as the most suitable way to present content (as also shown in Figure 7).

We consider the three following cases:

- cache miss: PC (palmtop) asking for a resource never requested before;
- raw content cache hit: PC (palmtop) asking for a resource already retrieved before by palmtop (PC);
- adapted content cache hit: PC (palmtop) asking for a resource already retrieved before by PC (palmtop).

Figure 8 presents performance results in the three cases above. More precisely, for each scenario, response time has been measured for both device types (PC and palmtop). The measurement campaign has been carried out by means of a network sniffer embedded in the client device. On the other hand, red columns refer to the expected response time computed on the basis of the theoretical model presented in the previous sections. Time units have been normalized by means of a proportionality constant making theoretical time equal to the actual response time in the cache miss case. Such constant has then been exploited in order to compute theoretical response times also in the other cases.

As it comes out from the picture, in both cases where caching comes into play the theoretical response time computed based on our model is very close to the actual measured values. This validates the pinball caching chart approach and allows concretely appreciating the benefits introduced by caching in real-world operational scenarios.

VII. Conclusions and Future Work

In this paper we have proposed a technique aimed at estimating the attainable performance improvement deriving from the application of a hierarchical caching technique to the operation of a system capable to deliver dynamically adapted content. The analysis focuses on the trend of the main parameters characterizing the system's behavior in the presence of caching. By applying such instrument to the tuning phase of the overall architecture we derive some useful insights. First, it comes out that it is worth devoting efforts to the optimization of the higher level adaptation mechanisms. The higher in the hierarchy the optimization strategy takes place, the more effective the attainable performance improvement. Though, the above result does not imply that little attention has to be paid to the lower layers of the architecture. Indeed, a high degree of reusability can be only achieved if low-level information is organized and represented in a thoughtful fashion. This clearly entails that a detailed analysis concerning effective data organization and representation techniques is carried out.

We just remark that the proposed technique actually represents a general means which can be exploited in order to estimate a system's attainable performance improvement in all cases where the system itself can be tuned through appropriate configuration of a well-defined set of parameters. The methodology proposed in this paper is inspired by an even more general approach allowing to clearly identify potential performance bottlenecks of component-based architectures [6].

References

[1] G. Armitage. Making the Internet go away. *IEEE Internet Computing*, 8(2):94–96, Mar.-Apr. 2004.

[2] A. Agostini, C. Bettini, N. Cesa-Bianchi, D. Maggiorini, D. Riboni, M. Ruberl, C. Sala, and D. Vitali. Towards highly adaptive services for mobile computing. *Mobile Information Systems*, pages 121–134, 2004.

[3] T. Laakko and T. Hiltunen. Adapting web content to mobile user agents. *IEEE Internet Computing*, pages 46–53, March-April 2005.

[4] J. Challenger, A. Iyengar, and P. Dantzig. A Scalable System for Consistently Caching Dynamic Web Data. IBM Research.

[5] Nilo Mitra (Editor). SOAP Version 1.2 Part 0: Primer. W3C Recommendation, June 2003.

[6] S. D'Antonio, M. Esposito, S. P. Romano, and G. Ventre. Assessing the scalability of component-based frameworks: the cadenus case study. *SIGMETRICS Perform. Eval. Rev.*, 32(3):34–43, 2004.