Apostolos Papageorgiou, Marius Schatke, Stefan Schulte, Ralf Steinmetz: *Enhancing the Caching of Web Service Responses on Wireless Clients*. In: Proceedings of the ninth IEEE International Conference on Web Services (ICWS 2011), July 2011, pages 9-16, ISBN: 978-0-7695-4463-2/11.
.

Enhancing the Caching of Web Service Responses on Wireless Clients

Apostolos Papageorgiou, Marius Schatke, Stefan Schulte, Ralf Steinmetz

Technische Universität Darmstadt
Multimedia Communications Lab - KOM
Darmstadt, Germany
{*apostolos.papageorgiou, marius.schatke, stefan.schulte, ralf.steinmetz*}*@kom.tu-darmstadt.de*

***Abstract*—Contrary to simple Web content, standard Web services do not offer their clients the possibility to use cached information without the risk that it may be out-of-date. This feature has not been worth its costs in realistic Web service usage scenarios until now. However, its absence may pose restrictions and impede possible benefits in a future scenario, where mediators are both willing and able to effectively minimize the amount of wirelessly transmitted data in the Internet of Services. This paper describes how developments in the Internet of Services start to motivate the automatic enablement of safe (i.e., always up-to-date) client-side caching for Web services. It presents our solution for generically adding this feature to any Web service, and, based on new experiments, reveals the limits beyond which the approach can offer significant benefits.**

***Keywords*-Web Services; Caching; Wireless; Adaptation;**

I. Introduction

Undoubtedly, enterprise systems have been the domain where Web services gained importance, being the most common technology for implementing Service-oriented Architectures (SOA) [13]. However, "everyday apps" might already be an equally important class of Web service consumers. Independently of which is the main application domain for Web services, the involvement of wireless devices as Web service consumers is increasing in both of them. A recent survey of TechTarget [7] positioned Web service-based mobile apps at the second place in the category "service-based implementations planned for the future" (planned by 60% of the questioned developers/companies), even higher than the "composite application assembly" (planned by 58%), which has been often named as the the main potential of SOA [13]. The popularity of Web services is due to the interoperability and platform-independence that are achieved through the self-description of the interfaces and the messages, but it is exactly this self-description that causes some communication overhead, for which Web services have been criticised since they appeared [5]. Enterprise systems may be affected by that only rarely but the same is not true for wireless devices.

Although some argue that the constraints of mobile devices (limited bandwidth, CPU, memory, or energy resources) are disappearing due to technological progress, the gap will not cease to exist. This is indicated by the latest analyses of future wireless communications. In the book of Sesia et al. [16] about LTE (Long Term Evolution of 3G mobile networks), five categories of user equipment are defined, with smartphones being placed only in the second or third category. According to this categorization, devices of higher categories will be able to use wireless internet connection rates up to six times greater than those of lower categories. Furthermore, the wired connections of the future will be even faster than that, not to mention the fact that devices less capable than smartphones, such as sensor nodes, will be able to consume Web services. So, the big differences in device capabilities and connection qualities will maintain the need for adaptation, as the size of the data that is processed and wirelessly transmitted is growing parallel to all other technological developments [1].

Therefore, most of the approaches that have appeared for reducing the overhead of Web service communication focus on wireless systems or are even specially designed for them. Client-side caching of Web service responses is a broadly used technique and many different algorithms and strategies exist for it. However, all client-side caching algorithms contain some risk of using information that is not up-to-date. If we want to be sure that our information is up-to-date, we have to send a new service request. Due to technical restrictions that will be explained in Section 3, (XML-based) Web services always transmit a complete (usually big) response when they receive a request. Thus, the following research question arises: "How can Web services exploit the caching concept, i.e., the reuse of information from former responses, but with certainty that they are up-to-date?". A mediator-based solution for enabling safe client-side caching of SOAP responses is described in Section 4 and evaluated in Section 5. To support a better understanding, related work is explored in Section 2, while the necessary background and an exact scenario are provided in Section 3.

II. Related Work

Much of the research effort in the area of caching has been devoted to the development of cache replacement strategies [14]. Such strategies concern the maintenance of the cache and their goal is to retain in the cache the entries that are most likely to be needed again soon, i.e., to maximize

the *cache hit ratio*. However, the hit ratio is irrelevant to the *freshness* of the cache information. Thus, if the usage of up-to-date information is of high importance, additional techniques have to be used in order to update the cache regularly. Such research is usually general-purposed and can be applied with small changes in many fields including web content (WWW caching), database information, and more. Some specialized Web service caching approaches have also appeared. These two classes of works (general-purposed and SOA-related) comprise our research landscape and their discussion is necessary in order to identify the challenge that has not yet been successfully addressed.

Although most of the caching approaches are based to some extent on the classical strategies "Least Recently Used" (LRU, cf. [9]) and "Least Frequently Used" (LFU, also cf. [9]), the research interest in the field is still alive. New approaches are still being developed, exploiting the characteristics of particular technologies in order to be more efficient. For example, Jelenković and Radovanović [8] recently published a replacement policy that has better performance than LRU and less complexity than LFU in the case of Zipfian request probabilities and big cache sizes. Cao [2] has enhanced cache management for mobile computing systems based on a concept for adaptively prefetching the content. While the mentioned solutions focus on the cache hit ratio, the state-of-the-art solution to optimize the cache freshness is the use of Invalidation Reports (IR) [3]. In that case, the servers indicate the changed data items to the clients at intelligently-determined intervals.

One of the most detailed analyses of the peculiarities of mobile caching for standard, XML-based Web services was provided by Terry and Ramasubramanian [17]. A demand for new technical solutions and standards, rather than algorithmic extensions, has been identified. Most of the challenges were related to technical enablements, and not to enhancements of algorithmic efficiency. Thus, the implicit directives provided by [17] led Elbashir and Deters [6] to the development of a transparent, cross-application cache for mobile clients, based on provider-metadata. Later, Liu and Deters [10] developed a new caching strategy for Web services, while Schreiber et al. ([15]) evaluated a middleware-based solution with caching and prefetching for mobile consumers of standard business processes.

Obviously, caching may be applied in different domains (databases, WWW content, Web services etc.), may have different goals (reduction of used bandwidth, reduction of user-perceived latency, reduction of server load, confrontation with connectivity loss etc.), and may be subject to different constraints (minimum hit ratio, minimum cache freshness levels etc.). In the domain of Web services, no existing solution can achieve the goal of *reduction of bandwidth and user-perceived latency* while simultaneously satisfying the constraint of *100% freshness*, i.e., always using up-to-date responses. The use of cached or prefetched responses without establishing a connection to the server each time leads, by definition, to a risk of using out-of-date information. When 100% freshness is desired, caching must simply be deactivated. The authors of [15] explicitly state that caching or prefetching of critical responses must be avoided. This statement is true, unless the responses are verified before use. This verification concept will be the basis of our solution and will be explained in the next section, following the description of our scenario.

III. Background and Scenario

A. Mobility Mediation in the Internet of Services

New service description specifications such as USDL [4], which include the business and operational aspects of services in addition to their technical details, turn Web services into perfectly tradable goods and lead to the so-called Internet of Services (IoS). In the IoS, a great number of Web services offered by different providers through global service marketplaces co-exist with a big set of clients with different features. As the trading and consumption of the services will have a loose relationship to their development, limited mobile clients will often need to use Web services that are not specially designed for them. For this reason, we are concerned in our project with the development of a Mobility Mediation Layer (MML) for the IoS. Complementary to other tasks (cf. [12]), such as automated inter-application context-enrichment, the MML supports proxied consumption of services that are available on the IoS-marketplace in order to perform overhead reduction for the wireless transmission. To achieve this, approaches like those analyzed in [11] are employed, along with further techniques.

Thus, the MML is also concerned with supporting wireless clients with the caching of service responses. How would traditional caching work? Naturally, every client can store responses for future usage. However, there are three main reasons why the client may avoid doing it:

- *Frequent changes*: The content changes so frequently that client-side caching does not make any sense.
- *Criticality*: The service call is so critical for the user that she/he needs to be 100% sure about the validity of the content, even if the old response has good chances of being up-to-date.
- *Legal issues*: Caching may be implicitly or explicitly stated to be illegal for a particular service.

In a global marketplace, it is difficult/unusual to know about these features, unless relevant information is included in a description such as USDL. But even if it is known that one of the above statements is valid, the only solution is to *always* request a new response.

B. The Challenges of Web Service Caching and the Vision of "HTTP-like" Caching

As already identified shortly after the appearance of Web service technologies, XML-based Web services present

many technical challenges and cannot be involved in a caching process similar to the one used for simple Web content (e.g., pure HTML) [17]. Based on directives provided by Web servers and supported by Web browsers, simple Web content is not reloaded each time it is requested, if the old (cached) content is still valid. The main technical reason why this mechanism cannot be transferred to the Web service technology is that Web services do not just rely on the HTTP-GET method for their communication, but they implement a rather more complex message exchange in order to export a diverse set of operations. As a result, Web services *always* transmit the complete response when they receive a request. The only alternative in order to support a validity check is to manually implement it inside the logic of each service. A survey among 20 services provided at *www.webservicex.net* (the 10 most popular and the 10 most recent) has revealed that not a single one of them has inherently implemented such a logic. The reason is obvious: the implementation effort outweighs the benefits, while the latter may often concern a limited number of clients. But what about a generic solution that could add this feature to any Web service in the IoS? Why has research not focused on it until now and what exactly would it enable?

In fact, the MML is one of the first systems confronted with a scenario where such a generic "HTTP-like" solution would be worth its effort, because:

- The MML has access to a huge amount of services over the global marketplace. Some of them would benefit from being enhanced with support for safe client-side caching. However, the MML cannot extend them directly with such a logic, because it has no access to their code or their hosting systems.
- The MML target group consists exclusively of wireless clients, while the service providers are often more concerned about larger PC-based clients.
- The MML can estimate better which services (or their clients) would profit more from client-side caching, because of extra information about cacheability that it can retrieve from USDL descriptions. However, a general-purpose mechanism should be able to function even without this extra information.
- The adaptation of Web services for mobility is the business model of the MML. Thus, the access either to the proxies generated by the MML or directly to the MML software in order to generate own proxies would be tradable goods in line with the MML business model.

For these reasons, the MML needs a mechanism which, when applied on any external Web service, allows the mobile clients to perform service calls such as the one shown in Fig. 1. Upon receiving a response with a particular code in its content (for example, 304, as in HTTP and in the example of Fig. 1), clients would know that their cached data were still

Example SOAP Request

```
<soap:Envelope>
……
<soap:Body>
       <getStreetNoiseStatus xmlns="……">
              <param1> Ocean Drive </param1>
              <param2> 33139 </param1>
       </getStreetNoiseStatus>
</soap:Body>
……
</soap:Envelope>
```

Example SOAP Response

```
<soap:Envelope>
……
<soap:Body>
       <getStreetNoiseStatusResponse xmlns="……">
              <incident>
                     <date> … </date>
                     <time> … </time>
                     <log> … </log>
                     <noiseRecordingInBytes>……
                     ……
                     ……</noiseRecordingInBytes>
              </incident>
              ……
              <incident>
                     ……
              </incident>
       </getStreetNoiseStatusResponse>
</soap:Body>
……
</soap:Envelope>
```

Envisioned next SOAP Response (at cache hit)

```
<soap:Envelope>
……
<soap:Body>
       <getStreetNoiseStatusResponse xmlns="……">
              <cachingCode> 304 </cachingCode>
       </getStreetNoiseStatusResponse>
</soap:Body>
……
</soap:Envelope>
```

Figure 1. Visualization of the vision of "HTTP-like" caching for Web services

up-to-date. This would not eliminate the need of establishing a connection, but could significantly reduce the wirelessly transmitted data. There is no way to be 100% sure of the validity of the data without establishing a connection each time that the data is needed, and this "100%" is a fixed goal of ours. Enabling request/response pairs such as these shown in Fig. 1 normally requires re-engineering and re-programming of the Web service. However, the MML needs a solution which somehow enables the described vision for any third-party Web service without access to its code and can be used by the clients without having to bother about any of the obstacles listed in the previous subsection (frequent changes, criticality, legal issues). Note that although the MML-scenario is used to explain our motivation and to support the technical understanding, the presented solution is general-purpose and may be very useful in other scenarios, as well.

IV. The Web Service Proxy Generator

The described vision is enabled by a software called Web Service Proxy Generator (WSPG). In addition to proxies for caching, the WSPG can also generate other proxies for overhead reduction (e.g., protocol-transformation, compression), but the latter are out-of-scope because they are based more on existing technologies, rather than on new concepts, techniques and evaluations, as is the case with caching. To provide a good understanding of the (caching) WSPG, an

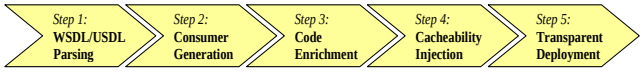

Figure 2. Workflow for the automatic generation of Web service caching proxies

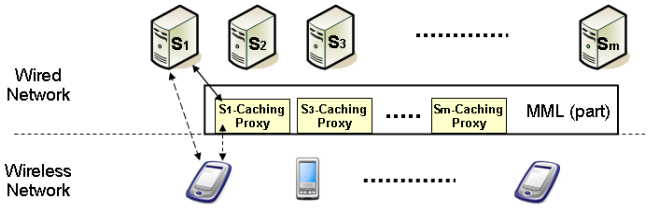

Figure 3. Overview of the technical landscape after some proxy generations

overview of its functionality is given along with a description of the resulting technical landscape. Then, a cost-benefit analysis is presented, which can also be used for deciding for which services the proxy generation is worth its costs.

A. Technical description and proxy generation logic

The WSPG is a module of the MML and should not be confused with the actual proxies that it generates, which are then deployed in a Web container of the MML. Fig. 2 shows the workflow with which the WSPG generates proxies based only on service descriptions, while Fig. 3 shows a possible technical landscape after the generation of some proxies. As shown in Fig 3, after the generation of a proxy, the direct communication of the consumer with the service (long dashed arrow) can be replaced with a proxied consumption, where the back-end call is performed between IT systems over fast connections (short direct arrow) and the wireless call (short dashed arrow) has the option of getting a very short answer, as envisioned in the previous section and shown in Fig. 1. In the following, the steps of the proxy generation are explained. Most descriptions are provided from an engineering point-of-view, though some helpful references to details of the java-based implementation are made. The highlights are to be identified in the conception of the workflow itself and in the innovative code adjustment techniques (esp. of Steps 3 and 4), but also in the difficulty of implementing such a logic generically.

Step 1 - WSDL/USDL Parsing: All the necessary information, such as operations, data types, interfaces, ports etc., are read from the service description. Because specialized information is needed, but also because the parser has to be extensible in order to support USDL, the WSPG does not use an off-the-shelf service description parser but a new one.

Step 2 - Consumer Generation: This step generates the part of the proxy that is responsible for calling the original Web service. For this purpose, the standard tool *wsimport* is transparently and automatically used by the WSPG.

Step 3 - Code Enrichment: The code generated by *wsimport* is different for each service. Each time, the WSPG has to dig deep into that code in order to find the points that have to be enriched. Then, it automatically modifies/adds code as needed. For example, (i) the addition of extra parameters in the request/response wrappers and (ii) the adjustment of the interface classes, which determine what the -new- WSDL (of the proxy) will look like, are two important code enrichment actions.

With regard to (i), a request (or response) wrapper is a class that determines what the SOAP request (or response) contains. After detecting these classes by analyzing the annotations of their code, the WSPG inserts the field *ifModifiedSince* to the request wrapper and the fields *statusCode* and *identTag* to the response wrapper, always preceded by the annotations that are necessary for their inclusion in an XML message and accompanied by the corresponding *getters* and *setters*. This allows for a communication pattern such as the one presented in Fig. 1 and for a validity check implemented based on the comparison of *ifModifiedSince* parameters with previously saved *identTag* parameters.

As for (ii), the WSPG has to detect the interface classes and replace pairs of RequestWrapper / ResponseWrapper annotations that are included there with the annotation @javax.jws.soap.SOAPBinding(parameterSytle=ParameterStyle.BARE). This allows for the use of extra parameters (see previous paragraph) without having to change the Web service signature, i.e., without additional wrapper classes for the new service.

Step 4 - Cacheability Injection: Similar code adjustments are also needed for the cacheability injection. However, this step concerns the actual handling of the cache, i.e., of the classes that access it or support it. Its logic is seperate from the previous step, and most of the results of the code enrichment actions of Step 3 are pre-conditions for the cacheability injection.

The most interesting actions of the WSPG during this step are the insertions of appropriate annotations to all the entities that must be persistable, i.e., are involved in the storage of request/response pairs, as well as the automatic generation of two further classes for each operation: A *builder* class and a *controller* class. The first is capable of storing request/response pairs so that validity checks can be performed, while the latter can search among the mentioned pairs and compare new responses with cached ones, in order to decide whether the new response or just a status code should be sent back to the wireless client.

Step 5 - Transparent Deployment: In this step, not only the code that results from the execution of Step 4 has to be packed and deployed but also the file structure and the configuration files of the Web container that will host the proxy have to be transparently adjusted, so that the generated proxy becomes automatically available as a new Web service.

B. Cost-benefit analysis

The proxy can be generated and deployed in up to a few seconds. Because the proxy is generated once a priori and not during a service call, this cost is trivial. However, there are other reasons why the generation of proxies for all known services may be impossible or undesirable. For example, the proxy-hosting platform may have limited capacity (it should be considered that a huge number of services shall exist on the IoS), the administrator may want to avoid having too many open ports or public interfaces because of security or management issues etc. Therefore, an estimation of the expected benefit should be supported by the WSPG.

The "total bandwidth saved by a proxy generation" (tbs) is exemplarily used here as the benefit metric because of its simplicity. Different metrics can be used, resulting in different models. If *hits* are the service calls for which the cached data can be used (i.e., 304 is returned) and *misses* are the service calls for which the complete response must be re-transmitted, then

$$tbs = avg\_bandwidth\_saving\_of\_hit \times number\_of\_hits - avg\_bandwidth\_loss\_of\_miss \times number\_of\_misses$$

which, if we define s_{avg} as the average response size, r as the hit ratio ($0 \leq r \leq 1$), s_{304} as the size of the minimal response that corresponds with a hit ($s_{304} >$ "size of the extra information needed in a proxy response"), and N as the total number of calls, gives after some simple calculations:

$$tbs \geq s_{avg} \times r \times N - s_{304} \times N$$

In order to assign values to these variables for particular services, the WSPG has two options. First, the expected values, e.g., for r or s_{avg}, could be read from an extended description such as USDL. Our related suggestions are being examined from the developers of USDL. Second, estimations can be performed based on information from standard descriptions. For example, the amount of parameters of an operation, as well as their types and name lengths, let the WSPG calculate the XML overhead needed to wrap the actual data in the corresponding responses. To provide a simplified example, n parameters (of primitive types) appearing sequentially with an average name length of l would produce in the response an overhead of $n \times (2 \times l + 5)$ characters, according to the schema `<paramName>...</paramName>`. Similar but more complex functions are used for calculating the overhead for the complete response, helping to estimate s_{avg}.

V. Performance Evaluation

In the following, the caching approach enabled by the WSPG will be thoroughly evaluated regarding performance. The presented approach can be combined with other overhead reduction techniques and its usefulness is not restricted to the cases in which the devices act as SOAP clients. Actually, in the MML it is currently often combined with REST-translation plus proxy-side caching, thus eliminating the external call completely when it is safe. The extra benefits of the presented approach are notable in these combined cases, as well. However, all other techniques are out-of-scope for our evaluation, whose purpose is to measure what our solution can contribute when used independently. Thus, all measurements refer to a scenario where devices, MML, and external services all communicate using SOAP, without any other optimizations.

A. Metrics and setup

Compared approaches: The presented approach is novel in that it achieves 100% freshness, so it should be compared with the only existent approach for achieving this, namely the direct call of the external Web service without the use of any cached results. However, the performance of our solution must also be compared with that of traditional caching, in order to evaluate what has to be sacrificed for achieving this absolute freshness. Thus, three approaches will be referred to, namely DCN (Direct Call, No cache), DCC (Direct Call with Caching), and PCV (Proxied Call with Validity check – our approach). The expectations are as follows: DCC performs best (but risks using old data), while PCV may complicate the communication compared to DCN (PCV uses extra data for the validity check and replaces one service call with two) but it reduces the wirelessly transmitted data in the case of a hit.

Dependent variables: We focus on two parameters, which – depending on the scenario – may be most important to the client: The reduction of the amount of wirelessly transmitted data or *saved bandwidth* (sb) and the *user-perceived latency* (upl).

Independent variables: The main variables that determine the performance are the *response size* (s) of the Web service and the *hit ratio* (r). For PCV, the hit ratio is the probability of the cached response being still up-to-date (see also IV.B), while for DCC it is, equivalently, the probability of using a cached response (of course, without knowing if it is up-to-date). s and r will be varied, while the rest will be controlled.

Controlled variables: Other parameters that affect the results are the client device, the network connection, the structure of Web service responses, and the workload (service call pattern). The next subsection explains how these variables have been chosen/controlled/varied and why.

Environment and simulated aspects: sb is exact and environment-independent. upl has been measured as response time plus parsing time. Parsing times have been measured on a real device, while response times have been measured with a simulator in order to ease tests with different networks (GPRS, UMTS, LTE). However, these response times have been often validated with sample tests from the device.

B. Setting the controlled variables

The measurements were done with an iPhone 3G (iOS 4.0.2) **device**. Less capable devices would obviously favour our approach even more. Furthermore, even if more capable devices hd been used, similar conclusions could be reached by slightly increasing the examined Web service response sizes, which would be completely realistic. Thus, the mentioned device was exemplarily selected because it is popular and it obviously does not favour our approach.

With regard to the **network connection**, different results will be shown, namely for GPRS (currently most used), UMTS (successor of GPRS in 3G networks), and LTE (future technology). EDGE and HSDPA perform similarly to GPRS and UMTS, respectively. WLAN can perform similarly to LTE, while the cases where WLAN achieves a performance almost equal to wired networks are obviously out-of-scope for any adaptation technique.

The experimental **Web services** have been chosen so that their response size can be easily varied. However, other service features may affect the results, as well. Responses with different structure may need different parsing times even for the same response sizes. To reflect this, two different real Web services have been selected: YellowMap's POI (Point-Of-Interest) service[1] (S_1) has a high parsing complexity, while a SOAP implementation of Apple's "iTunes new releases" service[2] (S_2) is easy to parse. Fig. 4 shows the parsing times measured for different response sizes and the *lines of best fit* that represent the parsing time as a function of response size s. The latter functions were calculated to be approx. $8.6228 \times s + 13.849$ and $4.632 \times s + 22.264$ for S_1 and S_2, respectively, while other Web services would probably give a line somewhere between them. Each point depicted in Fig. 4 is the average value after ten repetitions of the experiment. The deviation has been insignificant.

Different **workloads** have been tested. However, many of the results are presented for a scenario where a number of clients just send the same request twice. The number of clients is irrelevant because the saved bandwidth and the user-perceived latency reduction will be presented relatively. The existence of many clients is only assumed (and simulated) in order to assign all possible values to the cache hit ratio. More precisely, the whole response is always fetched at the first request, while the second request leads either to re-transmission of the response (cache miss) or to the transmission of our "small" 304 response. Thus, the enhancements cannot exceed 50%. This generic scenario was chosen because its triviality increases the comprehensibility of the results, its results consist again a bad case (minimal repetition of identical requests) for our approach, and no other scenarios are commonly accepted as realistic (see also [15]). This workload will be called 50-50, indicating that 50% of the calls have a chance to –but will not necessarily– be satisfied by cached results. The performance limits of the compared approaches will be illustrated independently of this scenario, as well.

[1]http://www.yellowmap.de/Partners/XML/PoiXmlServiceV21.asmx?wsdl

[2]http://ax.phobos.apple.com.edgesuite.net/WebObjects/MZStore.woa/wpa/MRSS/newreleases/rss.xml

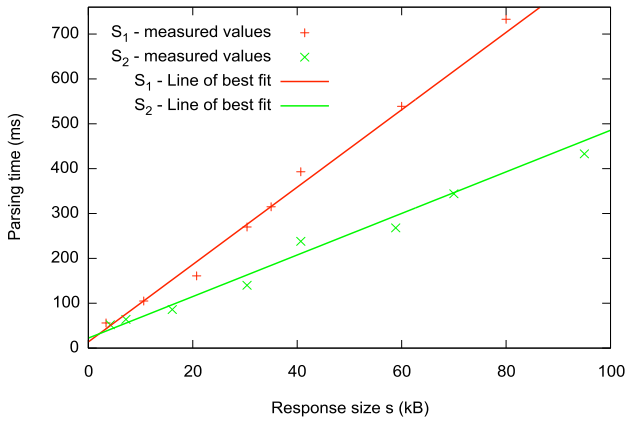

Figure 4. Examining Web services with regard to their parsing complexity

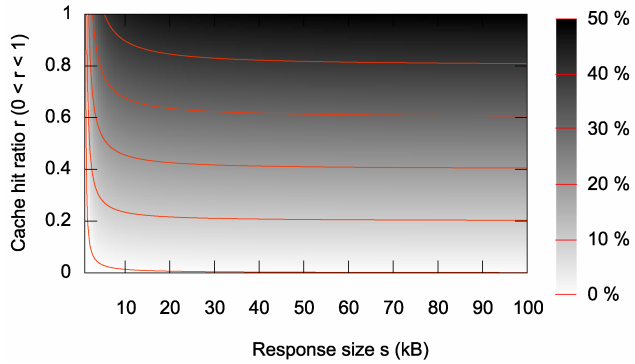

Figure 5. Saved bandwidth as a function of response size and cache hit ratio

C. Results

The explicit goal of PCV is to increase the saved bandwidth (sb). Both response time and parsing time reduction are actually caused by sb, so that upl depends up to an extent on sb. The question with regard to sb is for what values of s and r it becomes significant. Obviously, PCV cannot save much bandwidth when the exchanged messages are small and it is not worth it either when the expected cache hit ratio is low. For the 50-50 workload, Fig. 5 represents sb in percentage of the originally used bandwidth, i.e., the bandwidth used in the case of DCN. The results prove that sb gets significant values early, e.g., ca. 25% of the bandwidth can be saved for mediocre values of r even when s is smaller than 10kb. Note that sb is service-independent. The extra data used by PCV are normally trivial and may only cause a minimal performance degradation for very small values of

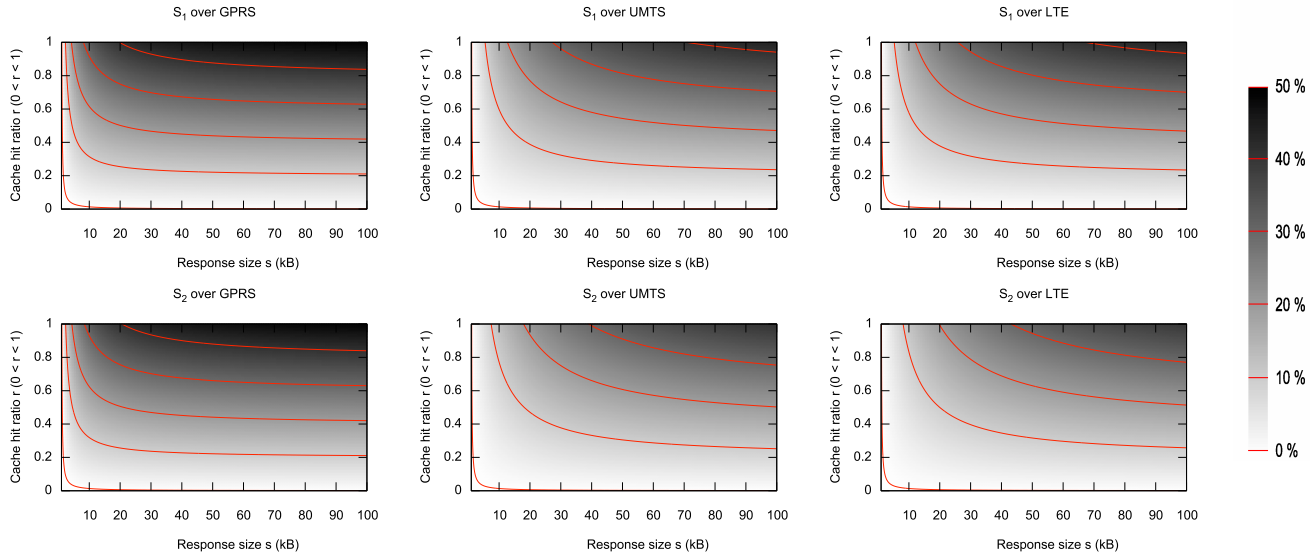

Figure 6. User-perceived latency reduction in the 50-50 workload (maximum reduction $< 50\%$) as a function of response size and cache hit ratio

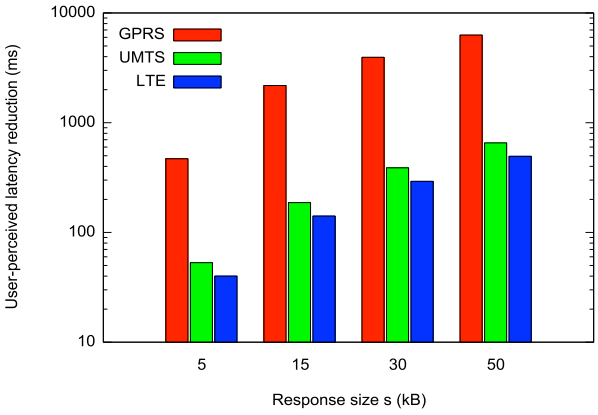

Figure 7. Absolute reduction of user-perceived latency for S1

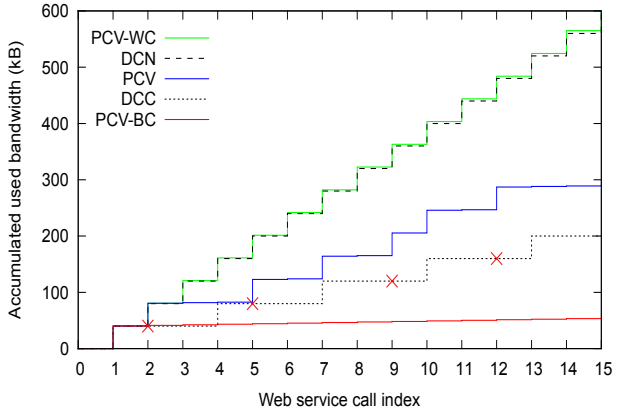

Figure 8. Illustration of the limits of the different caching approaches

s and r (under the lowest delimitating line in Fig. 5). In all other cases, the bandwidth savings are positive.

Fig. 6 shows the reduction of upl with a similar representation, i.e., as a percentage of the values of upl without caching (DCN). As expected, the reduction of upl is bigger for S_1. Interesting is the fact that, depending on the relationship between parsing times and response times, the percentage of upl reduction may be similar for LTE and UMTS (compare S_1-UMTS with S_1-LTE in Fig. 6). This is due to the fact that the parsing time sometimes dominates over response time in the overall upl, so that the connection becomes less important. The absolute upl reduction is, of course, bigger for UMTS, but the same does not necessarily hold when examining the relative reduction in percentage. Most important is the fact that upl presents significant reductions even though the call is proxied and the backend (wired) call is, naturally, also included in the measurements. Fig. 7 shows exemplarily and scenario-independently some absolute values for the upl reduction when S_1 is called with PCV (compared to DCN). A logarithmic scale is used because the reduction is much bigger for a GPRS network. Nevertheless, the upl reduction is significant for UMTS and LTE, as well. Again, each measurement has been repeated ten times and only average values are presented.

DCC has not been involved in the previous comparisons because the fact that no service call actually happens at a cache hit makes it senseless to talk about upl and lets the reader easily assume how the results would look like. However, in order to illustrate the spectrum between the performances of DCC and DCN where PCV is expected to

lie, Fig. 8 shows the used bandwidth for all three approaches in a particular setting. The goal here is a behaviour analysis, while the exact values are less important. Thus, Fig. 8 shows the accumulated bandwidth used by DCN, DCC and PCV for 15 periodical identical requests of a service whose response size is equal to 40kB. DCC uses for its cached objects a Time-To-Live (TTL) such that the objects expire after a time that corresponds with 3 calls, so that the 4th call has to fetch a new response. For PCV, the Worst Case (PCV-WC: no response is identical to a previous one) and the Best Case (PCV-BC: the response never changes) are drawn, showing the limits of the approach. As can be seen, for $TTL < \infty$, PCV can even achieve better results than DCC. However, this can only happen in cases in which the responses change rarely and, in such cases, DCC would not use many outdated information, anyway. With regard to the Worst Case, it is obvious that the difference to DCN is trivial. In addition to these limits, another case of PCV is depicted, where its hits and misses happen randomly (probability = 0.5). This last case achieves a good bandwidth usage, while the crosses denote calls for which DCC would have used outdated data. Although it would be interesting to examine where the PCV line appears in real scenarios, this issue is left open because a general view was desired and because no real data that are commonly accepted as realistic have been available.

The response sizes used throughout the evaluation were realistic and not very big. Both test services can give response sizes $> 500kB$, while other Web services cause even "heavier" communication. Frequent usage of such services causes significant amounts of data to be exchanged.

VI. Summary and outlook

After discussing why research for the automatic enablement of Web service caching with 100% freshness is now motivated by developments of the IoS, a solution has been presented for adding this feature to third-party Web services based on the idea of validity checks. It was shown that this solution can enhance the performance of wireless Web service calls, has unique features and can be considered as the most appropriate in particular scenarios.

Future experiments are planned in two main directions: Firstly, the solution should be evaluated for real Web service call traces in order to examine its performance in more realistic scenarios. Secondly, the reduction of the used badwidth is not necessarily the most important factor regarding energy consumption of mobile phones, which depends rather more on the time during which the device remains at a "high power status". Thus, the investigation of the circumstances under which our solution can also help reduce energy consumption is an important future goal.

VII. Acknowledgements

This work was funded in part by means of the German Federal Ministry of Economy and Technology under the promotional reference 01MQ09016 (Green Mobility Project). The authors take the responsibility for the contents. Many thanks to Philippe-Henri Marcy and Mouhannad Akhkobek for participating in the WSPG implementation, as well as to the authors of [15] for the knowledge exchange.

References

[1] C. Canali, M. Colajanni, and R. Lancellotti. Performance Evolution of Mobile Web-based Services. *IEEE Internet Computing*, 13(2):60–68, 2009.

[2] G. Cao. Proactive Power-Aware Cache Management for Mobile Computing Systems. *IEEE Transactions on Computers*, 51(6):608–621, 2002.

[3] J. Cao, Y. Zhang, G. Cao, and L. Xie. Data Consistency for Cooperative Caching in Mobile Environments. *IEEE Computer*, 40(4):60–66, 2007.

[4] J. Cardoso, A. Barros, N. May, and U. Kylau. Towards a Unified Service Description Language for the Internet of Services: Requirements and First Developments. In *International Conference on Services Computing (SCC '10)*, pages 602–609. IEEE, 2010.

[5] D. Davis and M. Parashar. Latency Performance of SOAP Implementations. In *IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '02)*, pages 407–412. IEEE, 2002.

[6] K. Elbashir and R. Deters. Transparent Caching for Nomadic WS Clients. In *IEEE International Conference on Web Services (ICWS '05)*, pages 177–184. IEEE, 2005.

[7] C. Frye. SOA Growth and Change. Available online at SearchSOA.com, March 2009. Last visited: January 2010.

[8] P. Jelenković and A. Radovanović. The Persistent-Access-Caching Algorithm. *Random Structures and Algorithms*, 33(2):219–251, 2008.

[9] R. Karedla, S. Love, and B. Wherry. Caching Strategies to Improve Disk System Performance. *IEEE Computer*, 27(3):38–46, 1994.

[10] X. Liu and R. Deters. An Efficient Dual Caching Strategy for Web Service-Enabled PDAs. In *ACM Symposium on Applied Computing (SAC '07)*, pages 788–794. ACM, 2007.

[11] A. Papageorgiou, J. Blendin, A. Miede, J. Eckert, and R. Steinmetz. Study and Comparison of Adaptation Mechanisms for Performance Enhancements of Mobile Web Service Consumption. In *IEEE World Congress on Services (SERVICES '10)*, pages 667–670. IEEE, 2010.

[12] A. Papageorgiou, B. Leferink, J. Eckert, N. Repp, and R. Steinmetz. Bridging the Gaps Towards Structured Mobile SOA. In *The 7th International Conference on Advances in Mobile Computing and Multimedia (MoMM '09)*, pages 288–294. OCG - ACM, 2009.

[13] M. P. Papazoglou and W.-J. Heuvel. Service Oriented Architectures: Approaches, Technologies and Research Issues. *The VLDB Journal*, 16(3):389–415, 2007.

[14] S. Podlipnig and L. Böszörmenyi. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4):331–373, 2003.

[15] D. Schreiber, E. Aitenbichler, A. Göb, and M. Mühlhäuser. Reducing User Perceived Latency in Mobile Processes. In *IEEE International Conference on Web Services (ICWS '10)*, pages 235–242. IEEE, 2010.

[16] S. Sesia, I. Toufik, and M. Baker. *LTE, The UMTS Long Term Evolution: From Theory to Practice*. Wiley Publishing, 2009.

[17] D. Terry and V. Ramasubramanian. Caching XML Web Services for Mobility. *ACM Queue*, 1(3):70–78, 2003.