A Semantic-Based Dual Caching System for Nomadic Web Service

Panpan Han, Liang Chen, and Jian Wu

Zhejiang University, Hangzhou, Zhejiang, China
ronson@zju.edu.cn

Abstract. As mobile devices become more widely used, they will emerge as a standard platform for hosting Web Service clients. Since mobile devices tend to be connected via a wireless network, they have to work with significantly less and fluctuating bandwidth as well as sudden and unexpected loss of connectivity. Moreover, mobile devices have other constraints, such as limited CPU, memory, or energy resources. To handle the above problems, we proposed a dual caching architecture and development method of web services for mobile devices. With the caching system between the client and server, we can cache the computing result and data downloaded from the server, which can reduce the latency and network load for workloads of web service. Meanwhile, the client side caching system can store the upload and download data.

Keywords: web service, caching, mobile devices.

1 Introduction

Currently, the Web services are spreading widely throughout the world. Web Services are an enabling technology for interoperability within a distributed, loosely coupled, and heterogeneous computing environment. The W3C defines a "Web Service" as "a software system designed to support interoperable machine-to-machine interaction over a network". It has an interface described in a machine-processable format (specifically Web Services Description Language, known by the acronym WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

The "standard" Web Services scenario assumes service providers and consumers as static and connected entities. But as mobile devices become more resource-rich and pervasive this is bound to change. As mobile devices become widely used, they are emerging as a means for hosting applications that consume Web Services(shown in Fig.1). However, since mobile devices tend to be connected to the Internet via very different kinds of networks, such as wireless LAN(802.11b), cellphone network(WAP), broadband network(cable modem), or local area network(Ethernet), they have to work with significantly less and fluctuating bandwidth as well as sudden and unexpected loss of connectivity. While fluctuations in the available bandwidth impact only the transmission speed of SOAP messages, the total loss of connectivity is more serious since it interrupts all SOAP traffic. Consequently, the use of mobile devices will introduce the novel notion of a nomadic Web Services participant that can suddenly disappear from the network and reappear at a later point in time.

The growth in the number of Web services has been phenomenal, hence, applying changes to existing Web services is impractical. For the same reason, the solution should be scalable and general enough to apply to all Web services. A good solution to improve availability of Web services should be transparently deployable and generally applicable[1]. Transparent deployment means that the solution must not require changes to the implementation of the Web services, either to the server and client side modules or to the communication protocol between them. Caching satisfies both the required characteristics of transparent deployment and general applicability. Caches are transparent to both the client and server components of the Web services.

This paper focuses on how to support nomadic Web Services clients by the use of dual caching system. With the combination of service characteristics and requirement of service, we can get cache semantic description(CSD). CSD is a XML which can show the type of the service operation(read or write), the service response characteristics and so on. Then the cache system can choose the corresponding caching strategy. The paper is structured as follows. Section three focuses on the novel dual SOAP caching system. An empirical evaluation of the dual caching system is presented in section four. A discussion of the results and a summary and future work section conclude the paper.

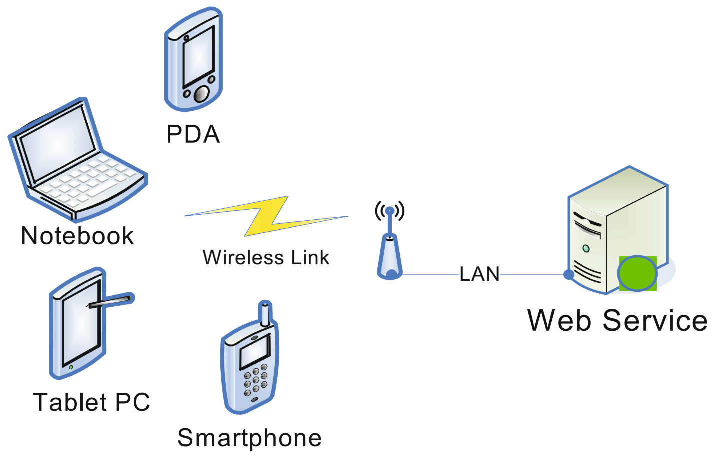

Fig. 1. Examples of mobile devices

2 Related Work

The concept of a cache [2] was first introduced for processor-memory communication; the concept then spread to file systems, computer networks, database systems and distributed object systems. Caches have been used to overcome disconnectivity, to decrease the latency of responses, and to increase throughput. A cache requires the identification of two cache semantics. These are cacheability and consistency maintenance.

There is much research on software caches in client-server architectures, with many objectives ranging from providing transactional guarantees, through providing consistency based on relaxed consistency models, to providing availability (e.g., [3], [4], [5] - just to name a few). It should be noted that although the hardware for mobile devices is improving at a tremendous rate, mobile devices still have limitations in terms of memory, processing power, communication bandwidth, and, in particular, power consumption. Consequently, caches targeted to mobile devices, in general, are smaller and simpler than caches targeted to servers or desktops.

A variety of systems have employed caching on mobile devices in support of disconnected access to files, databases, objects, and Web pages[6]. With the popularity of web service, several researches have been done on the caching of web services. [7] proposed a "HTTP-like" caching system. Upon receiving a response with a particular code in its content (for example, 304, as in HTTP), clients would know that their cached data were still up-to-date. This method reduces the wirelessly transmitted data to some extent. However, it would not eliminate the need of establishing a connection each time we need data.

To reduce the latency perceived by the user, [8] present a caching architecture for web services and an adaptive prefetching algorithm. The key characteristics of their approach are the compatibility with major mobile browsers and the independence of the caching proxy from the front-end application and the back-end services.

[9] introduces a dual caching approach to overcome problems arising from temporarily loss of connectivity and fluctuations in bandwidth. [10] present Differential Caches, with the accompanying Differential Updates method and the Mobile SOAP (MoSOAP) protocol, to avoid transfer of repeated data, sent by a web service to an application. The protocol is flexible in that other optimization techniques, such as encoding, can also be applied.

3 Dual Caching System

3.1 System Architecture

Fig.2 shows the architecture of the implemented semantic-based dual caching system. As mentioned in the previous sections, two transparent caches, one on the client side and one on the proxy side are required to overcome the loss of connectivity during SOAP traffic and to reduce the runtime overhead of mobile devices. The pair of caches is used to store the data sent by the web service to the invoking application and are managed by the respective client and server Cache Managers.

The Cache Managers controls all cache operations. The Cache Manager interfaces with the CSD preliminary to making cache decisions (e.g. read from cache, read from server side). The detailed description of caching strategy is described in the next chapter.

The proxy side cache(PSC) resides on a proxy server which has a reliable connection to the service provider. The proxy can alter the network traffic and can be a load balancing gateway, or a buffer to an intermittently available resource (e.g. WS).

The incoming requests from the client side cache(CSC) are sent directly to the proxy server. Responses from the service provider are first cached in the PSC and then sent to the CSC. In case the CSC can't be reached the PSC will wait for the CSC to issue a reconnect message upon which the PSC will resubmit all queued responses. To the client, the client side cache(CSC) appears as a local proxy residing on the mobile device.

When connected to network, the request sent by client side is intercepted by cache manager. Depends on the cache requirement and service semantic, we determine to return the client side cache data or to fetch response from server side. And when disconnected, for read operation, we also use CSD to determine whether to read data from cache or from server side. For write operation, we store the request data in the writer queue. If the client detect the network connected, we replay the request queue.

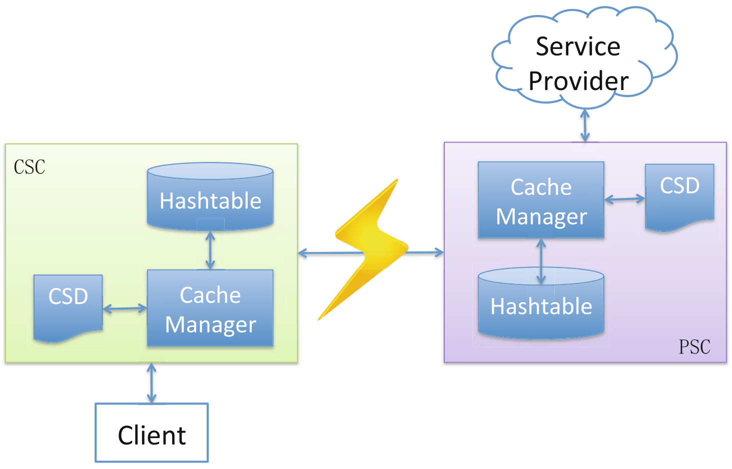

Fig. 2. The dual caching architecture

3.2 Caching Strategy

Cacheability of a request is the property stating that responses can be cached without the creation of an undesired program state [11]. Cacheability is always true for a request that is state-reading and non state-altering. The cached response, however, remains in cache until it is determined as invalid, or until the cache size is too large and the response was not recently used. A response is invalid if it has expired by an age value (e.g. HTTP time-to-live). An invalid response must be requested again from the live resource.

A cache is supported by strategies for maintaining equality between local responses and live responses. Consistency maintenance is the term used to describe these strategies. Hence, cached responses should closely resemble the responses of the live resource (e.g. WS). A strategy (e.g. implicit invalidation) removes or refreshes a cached response when a condition is true (e.g. response age is more than maximum age).

Requests which read the state of a resource are called READ requests (or READs). A READ request is synonymous with the terms query, state-reading, and state-dependent. Requests which alter (a.k.a. modify) the state of a resource are called WRITE requests. A WRITE request is synonymous with the terms update, and state-altering. A client receiving a local reply to a READ has performed a read-local operation. However, a client receiving a remote response to a READ has performed a read-through operation. A delayed state-alteration is known as a write back. However, an immediate state-alteration on the live resource is the result of a write-through operation.

Table 1 shows the attributes and corresponding values of CSD, and Fig 3 shows the key structure of CSD. In which the "type" attribute can be read, write and query, "response" attribute can be permanent, stable, predictable and random, "writeRequest" can be last-write-wins, context-free and other.

Server Side Caching Strategy. The server side caching strategy depends on the CSD, which is shown in table 2. And the write strategy for server side is No-Cache. No-Cache means all requests sent to proxy side, do not use cache.

Client Side Caching Strategy. The client side caching strategy is not only affected by service semantics but also has relation with service consistency requirement, which is shown in table 3 and 4. There are three consistency requirements: strong consistency means we must provide the newest response for each request, eventual consistency means after a period of time, we can return the newest response, and unconstrained consistency means there is no need to return the newest response for each request.

And the read strategy during connections is Automatic Update, the write strategy during connections is Write-through.

Table 1. CSD schema

Attribute Name	Attribute Value	Attribute Description
Name	Operation Name	the name of the operation
Type	Read	type of the operation: read operation, response is related to service state, operation doesn't change service state
Type	Write	type of the operation: write operation, response may be related to service state, operation changes service state
Type	Query	type of the operation: evaluate operation, response isn't related to service state but is related to parameters
Response	Permanent	feature of the response: permanent, response value is always the same for the same request value
Response	Stable	feature of the response: stable, response value is the same over a period of time for the same request value
Response	Random	feature of the response: random, response value is totally random
WriteRequest	last-write-wins	feature of the write request: latest write overwrites the previous ones
WriteRequest	context-free	feature of the write request: the order of write request has no effect to the response
WriteRequest	other	feature of the write request: different from the previous ones

Table 2. Server side cache read strategy

Service semantics	Permanent	Stable	Random
Server side caching strategy	Cache-Only	Passive Update	Automatic Update

Table 3. Client side cache read strategy during disconnections

	Strong consistency	Eventual consistency	unconstrained consistency
Permanent	Cache-Only	Cache-Only	Cache-Only
Stable	Failure	Cache-Only	Cache-Only
Random	Failure	Cache-Only	Cache-Only

Table 4. Client side cache write strategy during disconnections

	Strong consistency	Eventual consistency	unconstrained consistency
last-write-wins	Failure	LWW	LWW
context-free	Failure	Failure	Write-back
other	Failure	Failure	Failure

```
<complexType name="operationType">
      <sequence>
            <element name="resDepend" minOccurs="0"
                  maxOccurs="unbounded">
                  <complexType>
                        <attribute name="resName" type="string"></attribute>
                  </complexType>
            </element>
      </sequence>
      <attribute name="name" type="string"></attribute>
      <attribute name="type">
            <simpleType>
                  <restriction base="string">
                        <enumeration value="read"></enumeration>
                        <enumeration value="write"></enumeration>
                        <enumeration value="query"></enumeration>
                  </restriction>
            </simpleType>
      </attribute>
      <attribute name="responce">
            <simpleType>
                  <restriction base="string">
                        <enumeration value="permanent"></enumeration>
                        <enumeration value="stable"></enumeration>
                        <enumeration value="predictable"></enumeration>
                        <enumeration value="random"></enumeration>
                  </restriction>
            </simpleType>
      </attribute>
      <attribute name="writeRequest">
            <simpleType>
                  <restriction base="string">
                        <enumeration value="last-write-wins"></enumeration>
                        <enumeration value="context-free"></enumeration>
                        <enumeration value="other"></enumeration>
                  </restriction>
            </simpleType>
      </attribute>
    </complexType>
```

Fig. 3. Structure of CSD

4 Performance Evaluation

In order to evaluate the dual caching system, several experiments were performed. In the experiments Axis 2 was used as the Web Services middleware.

4.1 Experimental Setup

Web Service. The experimental WS implements methods supporting a game recommendation application. Users can get a list of recommended games, view game details and score the game. The WS implements the following Three methods, in no particular order:

– Get the game list (READ)
– Get the description of a certain game (READ)
– Give a game score (WRITE)

Metrics

– Mean Latency: the total time spent between a method call and receiving the method return
– Cache Hit ratio: the ratio of locally found responses

The client is instrumented to collect the metrics while it is subjected to a set of scenarios (e.g. Workloads). The client executes dependently on a request/connectivity pattern, and the metrics are collected after every request.

Workloads. The experimental workloads are a set of synthetic request patterns. A synthetic workload is a set of WS requests following the Zipf distribution. The requests are characterized by the ratio X (READ operations) to Y (WRITE operations), such that a workload is composed of X% READs and (100- X)% WRITEs. For all workloads, optimistic concurrency is assumed and an experimental run assumes no write-write conflict.

– P1; 100% READs, 0% WRITEs
– P2: 70% READs, 30% WRITEs
– P3: 40% READs, 60% WRITEs

Execution Environment. The experiments are executed within Samsung Galaxy Note 10.1 N8010 with android 4.1 OS. The Galaxy Note are configured to contain 2G Ram, and a Samsung Exynos 4412 Quad-core 1.4 GHz processor.

4.2 Experiment Result

To measure the impact of the dual caching system each experiment is performed with and without cache. Firstly, we measure the mean latency as the response message size changes from 1k to 100k with workload P1.

Fig.4 shows that the mean latency as the response message size changes from 1k to 100k. This figure shows a linear increase of mean latency as a function of the request message size. And it's obvious that with dual caching system the

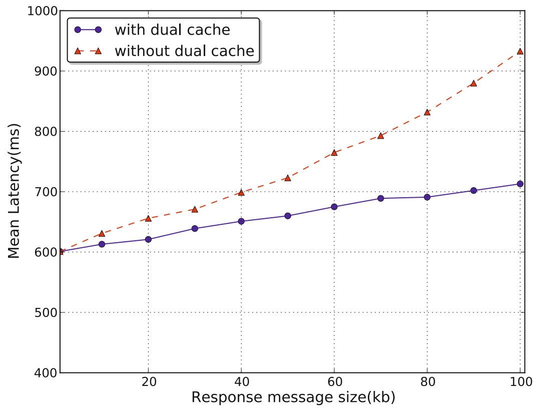

Fig. 4. Mean latency for workload P1 with cache and without cache

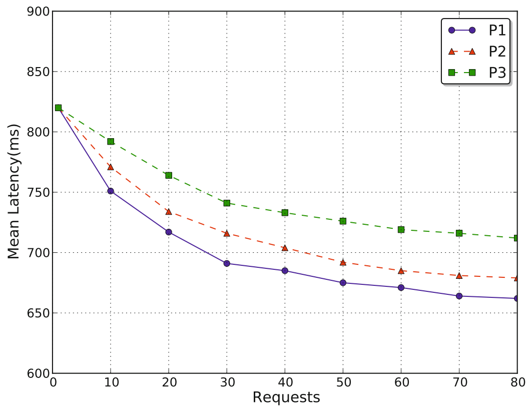

Fig. 5. Mean latency for different workloads

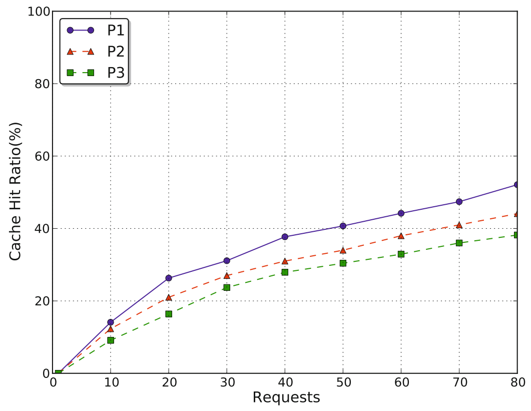

Fig. 6. Effect of Workload on Cache Hits

mean latency decreases notably. This is because the locally answered READ requests reduces the execution and network costs for the mobile device.

After evaluating the dual cache system with very basic workloads, we observe the effect of varying workloads on the latency of requests [Fig.5].

In the workloads P1-P3, the number of READ operations starts at 100% (in P1) and decreases by 30% for each workload (e.g. 40% in P3). The reduction in the number of READ operations is combined with a proportional increase of the number of WRITE operations. As a result, the mean latency decreases linearly with the decrease in number of READ operations in a workload.

This can be explained that many READ records are invalidated by WRITE operation. For example, before we score a game, the score is 87, which is cached in client hash table. After we score it by 70, the score will change and the cache is invalidated. So the system will fetch the score from the server side, which leads to the increase in mean latency.

Then we evaluate the cache hit ratio for different workloads. From P1 to P3, the number of READ operations decreases and the number of WRITE operations increases. From Fig.6 we can see the number of cache hits decreases linearly with the decrease in number of READ operations in a workload.

This can be explained by the same reason as the previous experiment, the WRITE operations caused many cache invalidation. So the the increase of the number of WRITE operations will lead to more cache miss and less cache hits.

5 Conclusions

This paper focuses on the challenges of enabling PDAs to host Web Services consumers and introduces a dual caching approach to overcome problems arising from temporarily loss of connectivity and fluctuations in bandwidth. Using a dual caching it becomes possible to handle loss of connectivity during the transmission of requests and the transmission of responses. An additional advantage of the dual caching is the reduction of latency and network load for workloads that contain significantly more read than writes.

Acknowledgement. This research was partially supported by the National Technology Support Program under grant of 2011BAH15B05, the National Natural Science Foundation of China under grant of 61173176, Science and Tech- nology Program of Zhejiang Province under grant of 2008C03007, National High- Tech Research and Development Plan of China under Grant No. 2009AA110302, National Key Science and Technology Research Program of China (2009ZX01043- 003-003).

References

1. Elbashir, K., Deters, R.: Transparent Caching for Nomadic WS Clients. Arch. Rat. Mech. Anal. 78, 315–333 (1982)
2. Barbara, D., Imielinski, T.: Sleepers and Workaholics: Caching Strategies in Mobile Environments (Extended Version). VLDB Journal 4(4), 567–602 (1994)
3. Garrod, C., et al.: Scalable query result caching for web applications. VLDB 1(1), 550–561 (2008)
4. Haas, L., Kossmann, D., Ursu, I.: Loading a cache with query results. In: VLDB 1999, pp. 351–362 (1999)
5. Oh, S., Fox, G.C.: HHFR: A new architecture for Mobile Web Services Principles and Implementations. Community Grids Technical Paper (2005)
6. Ramasubramanian, V., Terry, D.B.: Caching of XML Web Services for Disconnected Operation,`www.cs.cornell.edu/People/ramasv/WebServiceCache/WebServiceCache(techfest).pdf`

7. Papageorgiou, A., Schatke, M., Schulte, S., Steinmetz, R.: Enhancing the Caching of Web Service Responses on Wireless Clients. In: ICWS 2011, July 4-9, pp. 9–16 (2011)
8. Schreiber, D., Aitenbichler, E., Göb, A., Mühlhäuser, M.: Reducing User Perceived Latency in Mobile Processes. In: ICWS 2010, pp. 235–242 (2010)
9. Liu, X., Deters, R.: An efficient dual caching strategy for web service-enabled PDAs. In: ACM Symposium on Applied Computing 2007, pp. 788–794 (2007)
10. Qaiser, M.S., Bodorik, P., Jutla, D.N.: Differential Caches for Web Services in Mobile Environments. In: ICWS 2011, July 4-9, pp. 644–651 (2011)
11. Friedman, R.: Caching Web Services in Mobile Ad-Hoc Networks: Opportunities and Challenges. In: Proceedings of the Second ACM International Workshop on Principles of Mobile Computing, pp. 90–96 (2002)