Using a Cloud-Hosted Proxy to support Mobile Consumers of RESTful Services

*Shomoyita Jamal, Ralph Deters

Department of Computer Science, University of Saskatchewan, 110 Science Place, Saskatoon SK S7N 5C9, Canada

Abstract

While mobile handsets have always been heavily used for data communication most notably SMS and basic internet services like email and browsing, they are now beginning to be used increasingly often as platforms for accessing IT resources exposed as web services. However due to the resource constrains of smartphones and the often unreliable wireless connections it remains challenging to link them directly to web services. To overcome this challenge Mark Baccue proposed the idea of Mobile Cloud Computing, which is based on the concept of using cloud-hosted software components to aid advanced mobile handsets like smartphones in accessing web services.

This paper focuses on use of a cloud-hosted proxy as an effective way to support mobile consumers of RESTful web services. By linking the caching components of mobile devices with a cloud-hosted proxy it becomes possible to share caches and achieve significant performance boost for mobile consumers of RESTful web services.

Keywords: REST; Smartphone, Mobile Cloud Computing

1. The rise of Smartphones

Apple’s OMP (Original Newton Message Pad, 1993), a simple PDA, and IBM’s smartphone Simon (1992) were the first commercially available handheld computing devices that promised users time and location independent access to data and basic applications. Despite being commercial failures, they succeeded in transforming the public perception of computing and thus created the market/demand for handheld computing devices. In many ways they are the predecessors of the now omnipresent feature/smartphones and tablets that form the fastest growing segment of computing devices. Mobile handsets (phones) are expected to increase [1] from 4.3B (billion) subscriptions to over 5.8B in 2013 and thus dwarf the numbers of PCs (desktop, laptop, netbook) that are expected to rise from the current 1.1B to 2B in 2015 [2]. The primary reason for the growth of PCs and mobile handsets are the still low saturation levels in Asia, Africa/Middle-East and Latin-America. Within Europe and North-America, that have high
saturation levels of mobile handsets, telcos are forced to focus on improving their data services and upgrade their 3G networks to 4G e.g. via LTE (all-IP, 100 Mbit download, 50 Mbit upload) or WiMax as a means to attract customers. The improved data transfer in the networks, in combination with significant subsidization, has helped popularize next generation smartphones and thus create demand for higher data transfer rates. Gartner [3] reports that smartphone sales continue to grow (27% increase in 2nd quarter 2009, 40M units sold) and predicts that they will obtain a 29% market share by 2014 (406M smart phones sold/year). Consequently a virtuous cycle has started in which telcos continue to improve their networks, attracting more smartphone users that in turn fuel the demand for network improvements.

While mobile handsets have always been heavily used for data communication, most notably SMS and basic internet services like email and browsing, they are now beginning to be used increasingly often as platforms for accessing IT resources exposed as web services. However due to the resource constrains of smartphones and the often unreliable wireless connections it remains challenging to link them directly to web services. To overcome this challenge Mark Baccue [6] proposed the idea of Mobile Cloud Computing, which is based on the concept of using cloud-hosted software components to aid advanced mobile handsets like smartphones in accessing web services. “Cloud Computing” is a broad term that encompasses applications/functionality delivered as services over the internet and the software/hardware infrastructure to host them [7]. This “utility computing model” (John McCarthy, 1961), allows individuals and organizations to purchase virtualized hardware (Infrastructure as a Service, IaaS), software platforms (Platform as a Service, PaaS) or applications/functionality (Software as a Service, SaaS) in a pay-as-you-go manner, comparable to the metered purchase of electricity, gas or water.

The key challenge in using cloud-computing to support smartphones is to design the cloud-hosted support components to optimally compensate for the intermittent connection and the limited computing resources of the handset. Obviously the possible techniques and solutions for mobile cloud computing are closely linked to the type of services the handset consumes. While historically WS* [8] style web services dominated in the past service-oriented computing, REST [9] style web services have gained momentum in recent years especially in the mobile and consumer space.

This paper focuses on the use of cloud-hosted proxies as an effective means for supporting the caching components of mobile consumers. The remainder of the paper is structured as follows. Section 2 investigates the 2 dominate paradigms of web services namely SOA and REST. This is followed by a review of cloud computing in section 3 and a discussion of how to support smartphone applications via a proxy in section 4. Section 5 discusses the design and usage scenarios of a cloud-hosted proxy and section 6 presents a preliminary evaluation of the proposed proxy-based architecture. The paper concludes with a summary and outlook on future research.
2. WS* versus REST

Approximately at the same time mobile computing devices started to gain traction in the market, the enterprise middleware paradigms began to change. In 1996 Gartner [10] introduced services as a new integration and development paradigm. Nowadays service tend to follow either a Service-Oriented Architecture (SOA) or Representational State Transfer (REST) pattern.

The SOA [10, 11, 12] paradigm is based on the notion of stateless communication, stateless services, uses a well-defined protocol stack (WS*) and tends to be very successful within the enterprise and B2B domain. SOA is based on well defined, loosely coupled services [11] as the basic building blocks of the system. Within SOA, services are computational elements that expose functionality in a platform independent manner and can be described, published, discovered, orchestrated and consumed across language, platform and organizational borders.

In SOA (figure 3), services are offered by service providers that register them with registries (e.g. UDDI). Service consumers (aka clients) discover at runtime service providers by simply queering the registries. Originally designed around CORBA, SOA gained momentum with the widespread adoption of the fast evolving SOAP web services and their protocol stack (WS*) around 2001. Within the realm of corporate systems, access control is often mandatory due to legal and operational requirements (e.g. protecting personal user data). The Enterprise Service Bus (ESB) pattern (figure 4) is the most commonly used approach for controlling in- and out-bound activities within service-oriented systems. By enforcing (e.g. via firewall settings) that all communication/messages that need to cross the boundaries of the service-oriented system have to pass through a bus (e.g. ESB) it is fairly easy to establish effective access control (e.g. via HTTPS).

While SOA/WS* continues to dominate in enterprise systems especially B2B it continues to lose ground in the consumer space especially within Web 2.0 applications where REST is dominating. The Representational State-Transfer (REST) approach, which was developed by Roy Fielding [9], offers a lightweight and very well grounded alternative to SOA. In contrast to the SOA/WS* model that is based on stateless interaction and stateless services, REST is based on the concept of stateless interaction and stateful resources. The interaction in a REST system typically follows a request/response pattern in which each side assumes that all information is contained in the request and response. Within REST the basic building blocks are

1. **Stateful Services** often called resources.

2. **Uniform resource identification** (e.g. URL, URN)
   While each resource must have at least one uniform resource identifier, it is most often desirable to have more.

3. **Uniform interface** with a small set of verbs that have clearly defined operational semantics.

4. The state of a resource is sent via a **Representation** to the clients.

5. **Stateless Communication** protocols are used in the requester-provider interaction.
Leonard Robinson [13] identifies within his 4-level web maturity model two patterns that have become popular within REST. The first is the basic CRUD [14] (Create Read Update Delete) approach that follows a basic data-centric style. CRUD has gained significant interest due to the easy mapping on HTTP verbs. Create, read, update and delete can be achieved via POST, GET, PUT/PATCH and DELETE.

The second pattern that is less widespread is the use of hypermedia controls which is often also called HATEOAS (Hypermedia as the Engine of Application State). Unlike the data-centric CRUD pattern, the HATEOAS pattern focuses on the use of embedding links into the responses of request. By offering links in its responses, the server in effect controls the application state of the client in a very effective manner. Consequently the use of hypermedia controls (HATEOAS) is the most advanced pattern within REST.

3. Cloud Computing

Eric Schmidt introduced cloud computing in a conversation with Danny Sullivan at a Search Engine Strategies Conference in 2006 by saying “data services and architecture should be on servers. We call it cloud computing – they should be in a "cloud" somewhere.” [15]. According to Armbrust et al. [7], today computing is accessible in a way where users can access the services based on their requirements like utility services such as electricity, gas, water, and telephone. Armstrong et al. [7] and Barnatt [16] define cloud computing as “the applications delivered as services over the Internet”. According to “Evans Data” [17], after conducting a survey with over 400 software developers to determine their perceptions of leading cloud space vendors and providers, EC2, GAE, IBM’s cloud and Microsoft’s Azure are the most popular and top list companies. Amazon Elastic Cloud Computing (EC2) provides virtual machines where users can choose different Operating Systems and hardware architectures to run on their virtual machines. Based on the level of abstraction, Vaquero [18] defines three major scenarios in cloud computing.

- **Infrastructure as a Service (IaaS)** refers to service that exposes the hardware resources to users. Amazon EC2 is a successful IaaS implementation in the market.
- **Platform as a Service (PaaS)** provides computational resources as high level application platforms. Google App Engine (GAE) is an example of PaaS.
- **Software as a Service (SaaS)** focuses on exposing software functions as services (i.e. WS).

While seen traditionally as a means for organizations to outsource IT infrastructure, cloud computing can also be used to provide smartphones with additional resources [6].

4. Using Embedded Browsers to link Smartphones to the Cloud

A fairly recent trend in the domain of smartphone application development is the blending of native and browser applications. Traditionally developers had the hard choice to opt for a native smartphone application or a browser-based application. Native applications tend to be very device specific, take longer to develop, are harder to update but ensure full access to all smartphones features (e.g. accelerometer, GPS, sensors). Browser-based applications on the other hand are device-independent, simpler to develop but suffer from not being able to access all the features of the smartphone. To give developers more choice the Blackberry and Android platforms pioneered the concept of customizable sandbox settings for embedded browsers within signed native applications. By allowing the programmers to relax the sandbox settings of their embedded browser they can develop browser-centric applications that enjoy the same full access to smartphone features as a native one. Obviously the use of embedded browsers allows developers to target multiple platforms since they can re-use much of their interface. However the most attractive element of the embedded browser approach is that it enables the **continuous and transparent update** (perpetual beta) of an application which in turn dramatically improves the experience for the user. Rather than
having users download the “new version”, the update is achieved by simply providing the app with an updated web site.

Like all browser-based solutions, embedded browser apps favour lightweight web compliant technology. While it is fairly straightforward to consume RESTful web service via Java Script, it takes considerable effort and resources to consume WS* (SOAP) web services. Consequently embedded browser applications favour RESTful web services as the means for connecting to external resources/services. An interesting but often overlooked aspect of REST is the ability to cache its responses to requests. Unlike WS*, REST imposes clear read/write semantics which allows the caching of the read-request results. As shown in figure 5, a cache that stores the responses of requests to RESTful web services can be linked to an application or set of applications and help boost the perceived performance of the application by minimizing web traffic. To assist the mobile device it is important to also use a proxy that is keeping its cache updated. A cloud-hosted proxy can easily keep its cache updated by frequently checking with HEAD request for state changes in the RESTful services and therefore provide with little latency cache updates to the mobile client.

![Fig. 5. Mobile Client & Proxy Cache](image)

## 5. Cloud-Hosted Proxy

Using a cloud-hosted proxy and connecting it to the mobile clients has many advantages. A cloud-hosted proxy can help with the issues of intermittent connectivity by shielding service providers from sudden disconnections of clients and can also easily check for state-changes in the RESTful services/resources. The easiest way to achieve this is by periodically sending a HEAD request and to compare the ETag header of the response with the stored data in the cache. If the ETag of the HEAD request differs, it is an indication that the resource state has changed. While periodically checking is fairly easy to accomplish it can introduce additional loads. Please note that in most cases there are other caches between the cloud-hosted proxy and the web resource and that frequent requests to a resource help in keeping all caches fresh.

A way to avoid frequent HEAD requests, is to require that resources must offer an RSS feed that can be used to check for the current state e.g. ETag value. However, the most resource saving approach is to for the proxy to post a callback to the RESTful resource. Such a callback can then be used by the resource in case its state changes. However, using callbacks requires the RESTful resources to implement additional functionality in order to cooperate with the proxy which seems not likely.

It is important to note, that if a proxy is used by multiple mobile clients, it can use the responses of one client to keep the caches of multiple clients fresh. As a result, the more clients a proxy serves the more it can re-use responses for the benefit of all. This is of course most interesting if a single user interacts with multiple mobile
devices. In such a case the user can choose to share her/his cache across all the devices and in effect use the proxy as a means to store user-state across N devices.

6. Implementation & Evaluation

To evaluate the use of cloud hosted proxies as a means for supporting smartphones, we implemented the architecture shown in figure 5 using Blackberry’s WebWorks framework and EC2 (Amazon) as a cloud provider. To simplify the evaluation we used the Blackberry 9550 Simulator as the mobile client. Using WebWorks, the bulk of the client application was implemented using JavaScript. The JavaScript has access to the cache component that was developed in Java and is accessible via the JavaScript-Java bridge. The cache component intercepts all HTTP requests and decided based on connectivity and available cache data, how to process them. If the cache-component does not have the requested data, then it will connect to the proxy for the response. Figures 6 and 7 are the examples of Java code for Cache Manager and JavaScript code used in the WebWorks app.

A proxy server was used as the intermediary between the Blackberry client and RESTful web services. The proxy was developed in Erlang to ensure maximum scalability and high dependability when serving mobile consumers. Figure 8 is a snippet the core Erlang proxy that was developed.
For evaluating the performance of the proposed architecture, three different setups were deployed.

- The first experiment (without cache) was conducted with a Blackberry WebWorks application that consumes directly (no proxy) using a stable high speed connection and no cache.
- For the second experiment (with cache) the Blackberry WebWorks application was extended with a cache component.
- In the third experiment (with proxy) we used the setup of experiment two but included a proxy.

In all three experiments the Blackberry 9550 Simulator was hosted on a 4GB with an Intel®Core™ i5 CPU 650 that ran at 3.20GHz. A gigabit Ethernet connection was used to link the machine to the RESTful services and proxy (hosted on the EC2). The services and proxy ran on different EC2 virtual machines that used Intel®Xeon® CPU E5506 at 2.13GHz and accessed 1.7 GB of RAM. The systems were evaluated based on different numbers of concurrent clients that each executed 20 read requests to a RESTful web service. Figure 9, 10 and 11 shows the graphs that are recorded to calculate the minimum, maximum, and average time required in seconds respectively for the 3 different experimental setups. Figure 12 shows the throughput in seconds for the services as the number of requests is changed from 1 to 20 for the 3 different models.

These graphs indicate the reliability and scalability of the setup used in experiments 2 and 3. Using a cache and a proxy increases the performance, reduces latency and decreases bandwidth consumption for consumers of RESTful web services. While the use of a cache is of course expected to reduce network use and latency, it was interesting to see how well it performed with RESTful web services.
7. Conclusions & Future Work

This paper presents cloud-hosted proxy servers as an effective way to support embedded browser apps on smartphones. While the idea of using cloud computing as a means for supporting mobile resource constrained users has been proposed back in 2009 by Baccue there is still very limited research on how to implement his idea. The approach presented in this paper is currently linked to the embedded browser frameworks of the Blackberry and Android platforms. However, in our ongoing research we already experimented with hosting the cache in form of JavaScript objects completely in the browser and thus avoiding the need for specialized frameworks. Our future work will focus primarily on using more realistic workloads for measuring the performance of our approach. In addition we also plan to investigate more closely the possibility of sharing cached data across multiple devices owned by a single user and concept of using the proxy as the store for user state.

References