



Cost Efficient RESTful Services Caching for Mobile Devices

Iyad Ollite, Dr. Nawaz Mohamudally
School of Innovative Technologies and Engineering
University of Technology, Mauritius
La Tour Koenig
Mauritius

Abstract: The use of mobile devices to access the Internet has increased considerably in the past years and the increasing popularity of smart phones allows users to use applications to access content remotely using web services. The most widely adopted web service architecture is RESTful services, which consists of HTTP requests and replies. The challenges faced by mobile RESTful services are high response time, high battery use and bandwidth use. The HTTP protocol was not initially designed for RESTful services and although HTTP compatible caching servers can be used to cache RESTful services, they are not optimized for that purpose. This work proposes a novel transparent 2-tier proxy architecture, which can be easily deployed and used by producers and consumers of RESTful services. This proxy architecture highly reduces bandwidth usage which brings a financial cost savings to the mobile user.

Keywords: RESTful services, mobile web service, proxy architecture, web caching, mobile proxy, performance analysis, cost

I. Introduction

With the increasing popularity of smartphones, there has been a change in the way people use their phones and the way business reach customers, namely through the use of 'apps'. Apps are basically software, which allow a user to perform a given purpose and are downloaded on his mobile phone. Different apps provide different functionalities ranging from social networking, gaming to medical advice or news. Some apps are designed to operate without the use of Internet, for example single player games while others cannot operate without the Internet such as newsreaders, social networking tools and messengers.

In between, there are apps, which use the Internet only to provide additional features like advertising, streaming videos, push notifications or to provide updates. A large number of apps that require data from a server do so using a request-response model. The client will create a connection and make a request to the server. The server will listen permanently for incoming connections and send responses. Once the client acknowledges the response, the connection will be closed. To implement such functionality, several techniques can be used including REST, XML-RPC and SOAP. In recent years, there has been an increasing use of REST (Representational State Transfer) based APIs and it is currently the most popular API among developers.

REST refers to an architectural model in which a set of rules is applied to components, connectors and data elements in a distributed system. A REST based API does not provide for a specific format in which request messages are sent or responses are received. Given its simplicity, REST is the most commonly used technique for web APIs and apps. When applied to a web service, and used by apps, a RESTful API will consist of a RESTful service is a series of HTTP request and replies. When applied to a web service, and used by apps, a RESTful API will consist of the following:

- Location of the server
- The location of each service to be provided
- The data type of the data requested (XML, JSON, Images)
- The specification of the data; as defined by the developer
- The HTTP method to be used (GET, POST, PUT or DELETE)

Basically, a RESTful service is a series of HTTP request and replies.

II. Issues with RESTful Services and mobile devices

The architectural constraint of being stateless of a RESTful web service can lead to performance degradation. RESTful web services are appropriate for mobile devices since with each call clients need to send all state information. Consequently, even if there are frequent disconnections with the server, a client can continue its request its normal flow of resources without the need to restart from the first request. The performance issues faced by RESTful services when called from mobile devices include:

A. Increase in network latency

For different requests, the client will generally initiate different connections to the server. That is, for each request, a new socket connection is required; although this part is usually handled by the operating system and

by libraries already available to the client when he is making an HTTP request. RESTful servers might use a keep alive mechanism and allow a client to reuse the same connection but these tend to be short-lived (about 180 seconds). So for any given request a new socket needs to be established to the server.

B. Increase in processing time

Since the server does not keep session information for a client, it will have to do more processing for a given request. For example, with each request, the server will have to recheck the authentication/credentials of the client before processing the actual request.

C. Increase in data transfer and financial cost

For RESTful services, the client manages his own session information. Consequently, for each request he has to send more information to the server. Additionally, all header data needs to be transferred over again to the server. These headers can be larger than the actual data requested required to process the call. Since resources need to be fine-grained, the client will need to make several requests to the server in order to get all required information. The server as well will need to transfer more information and header data with each request.

Mobile Internet is usually sold in subscriptions where users pay for amount of data they download. That is, the more they use mobile Internet, the more they have to pay and mobile broadband Internet is expensive compared to cable based broadband Internet such as ADSL.

III. Proposed RESTful Services Proxy Model

The proposed RESTful proxy system is a 2-tier model with one component implemented on the client device, referred as RESTful Proxy Client (RESTPC) and a server component referred as RESTful Proxy Server (RESTPS) to be used by the service provider. The aim of the model is to minimize the amount of bandwidth used and therefore bring a cost saving to the end user.

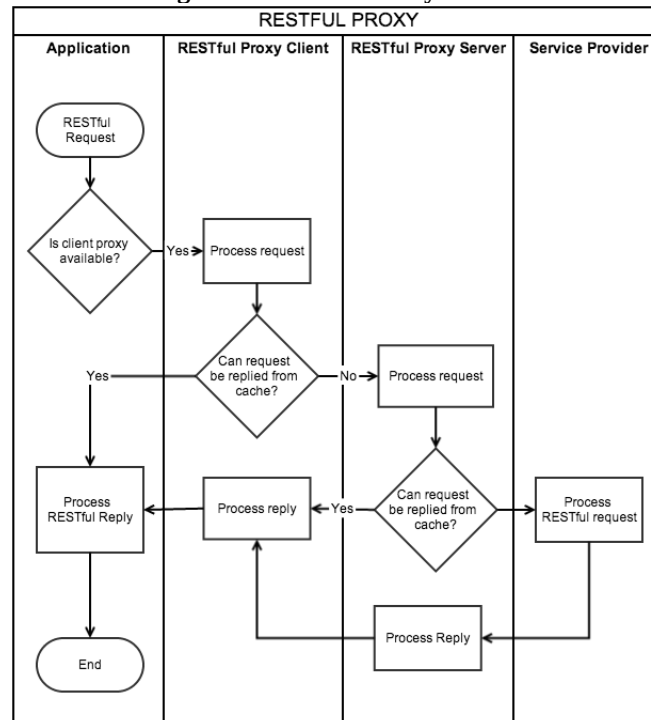
When using the RESTful proxy, all RESTful requests from the application are sent to the RESTPC which will analyze the request to determine whether the response can be provided from cache or needs to be fetched from the RESTful Proxy server. If available in the local cache, a RESTful response is sent to the application else a reformatted request is sent to the RESTPS which will process the request and check whether the response can be replied from its cache or needs to be fetched from the service provider. If available from cache, a custom formatted reply is sent to the RESTful proxy client. RESTPC will process the reply and convert it back to a complete RESTful response, which is then forwarded to the application. If the response was not available in the RESTPS, the RESTful request is reconstructed and sent to the service provider. When the RESTful reply is received, the RESTPS reformats the message and sends it to the RESTPC. The latter will rebuild the full RESTful response and send it to the application. When using this model, the application on the mobile device and server at the service provider receives unmodified RESTful requests and replies; the RESTful Proxy is transparent.

IV. Proposed RESTful Services Proxy Model

The proposed RESTful proxy system is a 2-tier model with one component implemented on the client device and a server component to be used by the service provider. The model assumes that the user installed the RESTPC on his device and that the application developer provided for an option to the end user to use the proxy. RESTPC listens for HTTP request on a given port as specified by the user. When an application makes use of the RESTful proxy service, the request is sent to the RESTPS which will process it. The RESTPS will listen for request on a given port and process all requests from the RESTPC.

If the RESTful proxy is unavailable the application will make the request directly from the service provider; else the request is forwarded to the RESTful proxy. The application will determine if the RESTful proxy client is running by checking if the service is running on a certain port. In the event the RESTful proxy is available, the request is processed as depicted in figure 1.

When using the RESTful proxy, all RESTful requests from the application are sent to the RESTPC. The RESTPC will analyse the request to determine whether the response can be provided from cache or needs to be fetched from the RESTPS. If available in the local cache, a RESTful response is sent to the application else a reformatted request is sent to the RESTful Proxy server. The RESTPS will process the request and check whether the response can be replied from its cache or needs to be fetched from the service provider. If available from cache, a custom formatted reply is sent to the RESTPC. The RESTPC will process the reply and convert it back to a complete RESTful response, which is then forwarded to the application. If the response was not available in the RESTPS, the RESTful request is reconstructed and sent to the service provider. When the RESTful reply is received, the RESTPS reformats the message and sends it to the RESTPC. The latter will rebuild the full RESTful response and send it to the application. When using this model, the application on the mobile device and server at the service provider receives unmodified RESTful requests and replies; the RESTful Proxy is transparent. Whether the RESTful proxy is used or not, the mobile application will send an HTTP request and receive and HTTP reply; the service provider will receive an HTTP request and send an HTTP reply.

Figure 1: RESTful Proxy Model.

V. RESTful Services Proxy Caching Techniques

Like all cache systems, the RESTful Proxy needs to address several issues in order to be as efficient as possible.

A. Cache placement

A client proxy resides on the user's mobile device and another proxy is installed on a machine on the same local area network or same machine as the service provider.

B. Cached content

Given the characteristics of a RESTful service, only GET responses are usually cached. However, the proposed model caches GET responses as well as headers of GET, POST, PUT and DELETE requests and responses. This is possible since the client proxy and server proxy have their own internal communication protocol and can reuse previous headers to rebuild a complete request to be sent to server and header responses to be sent to clients.

C. Inter-Cache communication

The communication protocol modeled allows the RESTPC to communicate with the RESTPS; RESTPS do not share content among themselves. In certain cases, the RESTPSC might send a request to the RESTPS which contains only an identifier to a previous request but the RESTPS might no longer have a record of the request. In that case, the RESTPS will reply a response with the status 301(Moved permanently) and The RESTPC will then need to send the full request. The identifier system implemented does not require the RESTPS to check if the RESTPC has a cached version of the content. Since the RESTPC sends the unique ID of the resource it has in its cache, the RESTPS can determine whether to send a new response or a response with status 304 (Not Modified).

D. Cache replacement policy for RESTPC

For the RESTPC, two separate replacement policy based on whether responses contain cache directives are used. Firstly, the Next To Expire algorithm (NTE) is used since it is relatively simple algorithm and therefore does not require much processing power and consequently, the amount of battery use is limited. However, for NTE to be efficient, the response headers must include the max-age directive found in the Cache-Control response header or the Expires header directive is specified. For responses not containing any cache directive, the Least Frequently Used (LFU) algorithm is used. LFU as well is a straight forward algorithm which does not require much processing power. In the absence of cache directive, the client proxy will still have to send the request to the server proxy to ensure that the content has not changed. To be able to keep track of which items is least frequently used, every time a resource is used from cache a counter associated to it is incremented.

E. Cache replacement policy for RESTPS

For RESTPS, For the server proxy, the NTE (when cache directives are available) and a Greedy Dual-Size (GDS) (when cache directives are not available) algorithms are used. The GDS used takes into consideration the time required to process a request by the server and the size of the response. This algorithm is different from the regular GDS algorithms in that the server proxy is really close to the service provider and the number of network hops between the service provider and server proxy is two or less. Rather than using network hops to determine the cost of a resource, the actual processing time of the request by the service provider is used. The second

parameter considered for the algorithm is the compressed size of the resource. The weight for a given resource is given by $1-(1/(\text{Cost} \times \text{Size}))$. The element with the least weight will be evicted from the cache.

VI. Implementation

A prototype above model has been implemented for mobile devices running the Android operating system; the RESTPS has been implemented in Java. The RESTPC will run on a certain port on the mobile device and the application can connect to that port. When coding the application, the developer needs to check if a service is running on that given port and if the port is accepting connections, he then requests that all calls are made using that proxy. For test purposes, the end user can install the software and configure its mobile device settings to use the RESTPC as proxy. The pseudo code to be implemented by the mobile developer is as follows:

```

Read global settings
If Restful Proxy Port defined
    Set Proxy Host as localhost for current application
    Set Proxy Port as defined in global settings for current application
Endif
Make standard HTTP request to call RESTful Service

```

RESTPS runs on a certain port and is installed on the same machine providing the service or on the same local area network. It is totally transparent to the developer and no changes are required to be made by the RESTful service provider developer. A RESTful news service was also implemented to determine the effectiveness of the system on a real service.

VII. Performance Review

In order to access the performance of the above described mobile caching architecture, the following metrics were used, namely hit ratio, byte-hit ratio and bandwidth usage. The following metrics were compared for the following architectures:

- RESTful service without
- RESTful service with a single web cache
- RESTful service with RESTful Proxy without the client proxy active
- RESTful service with RESTful Proxy with the client proxy active

For all the metrics, the RESTPC cache size tested was 100KB while the RESTPS cache size was 5MB.

A. Hit Ratio

The hit ratio represents the ratio of number of items fetched from the cache with respect to the total number for requests made. For the RESTful Proxy model, hit ratio is calculated at for both the RESTPC and RESTPS. Figure 2 shows the hit ratio of the different architectures. It can be seen that when using a single cache, the hit ratio is slowly increases to 72% as the number of requests increases and cache is filled. When using the RESTful Proxy both with and without the client cache, the hit ratio is 81.7%; that is an increase of around 9.5%. This increase can be explained by the cache replacement policy used by RESTPS. This increase can be explained by the fact that more resources can be stored in the cache when using compression and optimizing the data exchange between the RESTPC and RESTPC.

B. Byte Hit Ratio

Byte hit ratio represents the ratio of total size of resources that were fetched from cache with respect to the total size of resources served. Figure 3 shows the byte hit ratio of the different architectures. This metric is calculated at for both the RESTPC and RESTPS. When using a single cache, the byte-hit ratio is around 72%. When using the RESTful proxy, the byte-hit ratio is around 82%. The byte-hit ratio is higher due the Greedy Dual-Size algorithm implemented. By keeping larger but compressed resources in the cache, these larger items are served directly from cache.

C. Bandwidth Usage

Bandwidth usage represents the total number of bytes that was sent over the network and the bandwidth savings represents the total size of resources that were not sent over the network and retrieved from cache. Since the model proposed is a 2-tier one, when a resource needs to be validated from the RESTPS either the complete resource needs to be sent or only a validation message.

When no cache is used, the bandwidth used by the service provider and clients for the 1,000,000 requests is 5,130MB. When a single cache is used, the bandwidth used by the service provider falls to 1413MB; representing a bandwidth savings of 72% for the service provider. However, for the clients, 5,130MB of data is still transferred since the data, which is cached, will be served from cache rather than service provider.

When the RESTful Proxy is used without the client component, only 1,026MB is required for all requests. This represents the bandwidth consumed by both the server and clients; a reduction of bandwidth of 80%. This reduction of data usage can be explained by the use of compression by the RESTful proxy. When the RESTful proxy is used with both client and server caches, the amount of data transferred for all the 1,000,000 requests is only 712MB; a bandwidth usage reduction of 86%. The additional gain is explained by the fact that some requests can be served from the client's local cache and therefore no network data transfer is required.

Figure 2: Hit Ratio

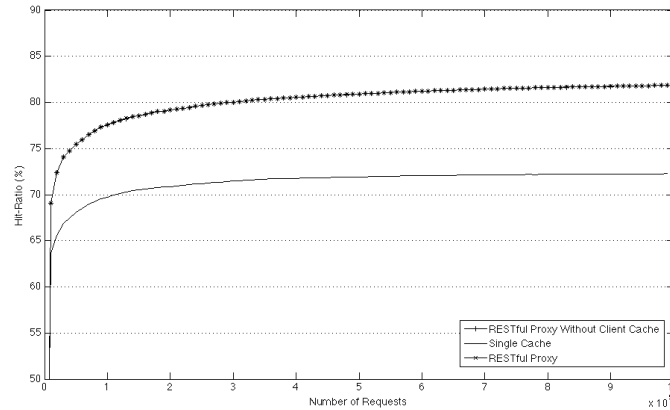


Figure 3: Byte Hit Ratio

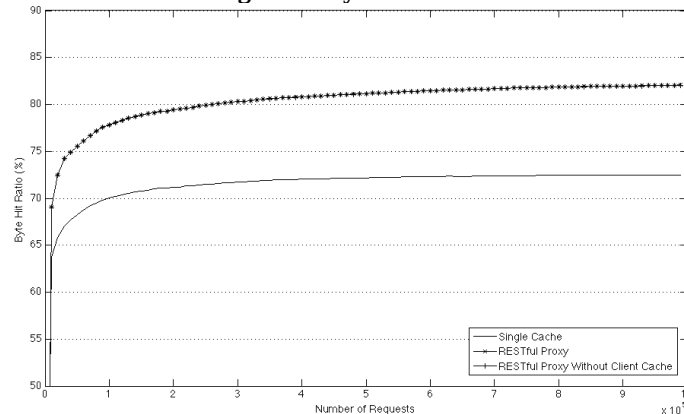
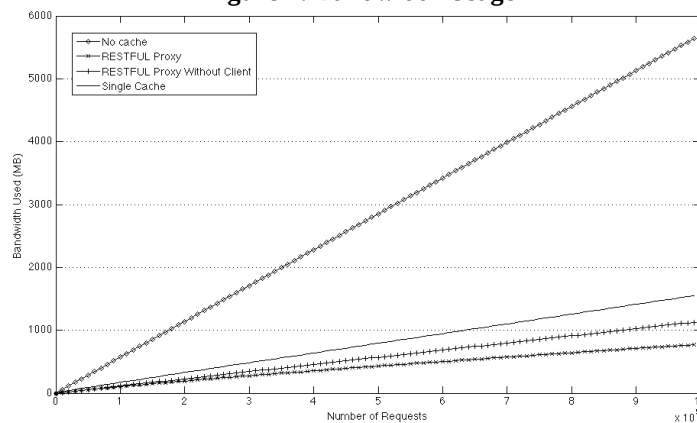


Figure 4: Bandwidth Usage



VI. Conclusion

Based on the above results it can be deduced that the RESTful Proxy will bring cost savings to both the service provider and mobile users. For mobile users, there is a reduction in bandwidth usage and therefore a reduction in the amount spent on mobile data package. For the service providers, less bandwidth is required as well as less CPU time. Consequently, the service will be able to serve more requests before reaching full capacity.

The proposed model also has its limitation. Firstly, it focuses on only the economic gain for the mobile user. A full study needs to be done to determine the impact of the system on other key metrics such as response time, network latency, processing time and battery use. Additionally, different compression algorithms need to be tested with the above model to determine which is most appropriate.

References

- [1] Hu, X. and Zincir-Heywood, A. N. 2005. Understanding the Performance of Cooperative Web Caching Systems. In Proceedings of the 3rd Annual Communication Networks and Services Research Conference (May 16 - 18, 2005). Communications Networks and Services Research Conference. IEEE Computer Society, Washington, DC, 183-188.
- [2] Rodriguez, P., Spanner, C., and Biersack, E. W. 2001. Analysis of web caching architectures: hierarchical and distributed caching. *IEEE/ACM Trans. Netw.* 9, 4 (Aug. 2001), 404-418.
- [3] Teng, W., Chang, C., and Chen, M. 2005. Integrating Web Caching and Web Prefetching in Client-Side Proxies. *IEEE Trans. Parallel Distrib. Syst.* 16, 5 (May. 2005), 444-455.
- [4] Bzoch, Pavel; Matejka, Lubos ; Pesicka, Ladislav ; Safarik, Jiri, 2012, Towards caching algorithm applicable to mobile clients, In Federated Conference on Computer Science and Information Systems (FedCSIS), pp 607 – 614.
- [5] Mwaffaq Otoom and JoAnn M. Paul. 2008. Holistic design and caching in mobile computing. In Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis (CODES+ISSS '08).
- [6] Santhanakrishnan, G.; Amer, A. & Chrysanthos, P. K. (2005), Towards universal mobile caching., in Vijay Kumar; Arkady B. Zaslavsky; Ugur Çetintemel & Alexandros Labrinidis, ed., 'MobiDE' , ACM, , pp. 73-80 .
- [7] J. Fernandez, A. Fernandez, J. Pazos,. (2005), Optimizing web services performance using cache., In International Conference on Next Generation Web Services Practices.
- [8] Dawar, M., Singh, C. (2013), 'Study on web Caching Architecture: A Survey', In International Journal of Advanced Research in Computer Science and Software Engineering., pp 581-585 .
- [9] Wang, J. (1999), 'A survey of web caching schemes for the Internet.', *Computer Communication Review* 29 (5) , 36-46 .
- [10] Athena I. Vakali , George E. Pallis (2001), A Study on Web Caching Architectures and Performance.
- [11] Catrein, D.; Lohrer, B.; Meyer, C.; Rembarz, R. & Weidenfeller, T. (2011), An Analysis of Web Caching in Current Mobile Broadband Scenarios., in 'NTMS' , IEEE, , pp. 1-5 .
- [12] Cai, X.; Zhang, S. & Zhang, Y. (2013), Economic analysis of cache location in mobile network., in 'WCNC' , IEEE, , pp. 1243-1248 .
- [13] Wathsala, W.V., Siddhisena, B., Athukorale, A.S. (2008), 'Next generation proxy servers.', In 10th International Conference on Advanced Communication Technology, 2008. ICACT 2008., pp 2183-2187.
- [14] Qian, F.; Quah, K. S.; Huang, J.; Erman, J.; Gerber, A.; Mao, Z. M.; Sen, S. & Spatscheck, O. (2012), Web caching on smartphones: ideal vs. reality., in Nigel Davies; Srinivasan Seshan & Lin Zhong, ed., 'MobiSys' , ACM, , pp. 127-140.