Network Data Buffering for Availability Improvement of Mobile Web Applications

Tomoharu Imai(✉), Kouichi Yamasaki, Masahiro Matsuda, and Kazuki Matsui

Network Systems Laboratory, Fujitsu Laboratories Ltd., 4-1-1 Kamikodanaka, Nakahara-Ku, Kawasaki, Kanagawa 211-8588, Japan
{imai.tomoharu,yamasaki.koichi,matsuda,kmatsui}@jp.fujitsu.com

Abstract. In recent years, we have seen an explosion in the use of smart devices. With several different competing OS platforms on these smart devices, developers have turned to web applications as an effective way to provide cross-platform services. However, because many web applications are designed to handle UI interactions locally on a user's device and to process data remotely in the cloud, it is difficult for them to continue running while offline. In this paper, we propose a data synchronization technology that buffers and minimize network communications to address problems associated with dropped network connections as well as low bandwidth and/or high latency environments. To test the technology's effectiveness, we applied it to some typical web applications and compared their performance in environments with dropped connections.

Keywords: Mobile web applications · Data buffering · Synchronization technology

1 Background

In recent years, we have seen an explosion in the use of smartphones, tablets, and other smart devices. These smart devices run a variety of operating systems, including Android, iOS, and Windows Phone. Applications depend on proprietary APIs and SDKs for each mobile operating system, all of which are incompatible with each other. The need to build a separate application for each operating system thus drives up development costs. On the other hand, each operating system also has a browser engine capable of rendering HTML5 content, allowing a single HTML5 web application to be run on a variety of different devices. For this reason, web application development for smart devices has become more popular in recent years.

As shown in Fig. 1, web applications save a variety of data on mobile devices; this includes audio/video data taken with a device's built-in cameras and microphones as well as business data downloaded from the cloud. Consequently, web application security is sometimes lacking. To address this, systems with stringent security requirements—such as those in enterprise businesses—have begun to adopt thin client systems that run applications in the cloud. However, thin client systems can send and receive a large volume of commands and screen data that may noticeably reduce an application's responsiveness on slower network connections (as shown in Fig. 2).

S. Sigg et al. (Eds.): MobiCASE 2015, LNICST 162, pp. 3–11, 2015.
DOI: 10.1007/978-3-319-29003-4_1

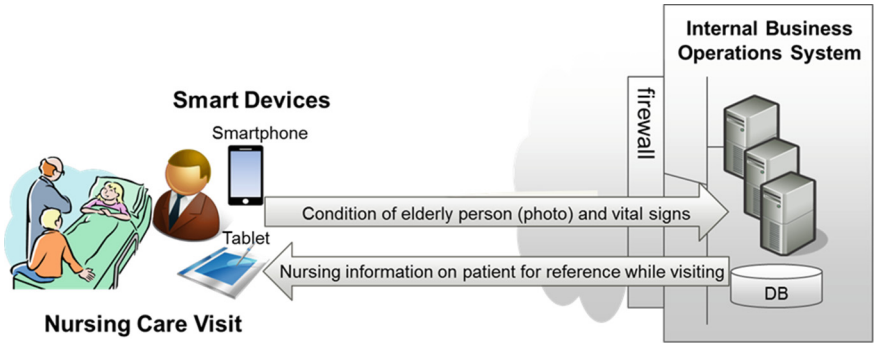

Fig. 1. Web application usage

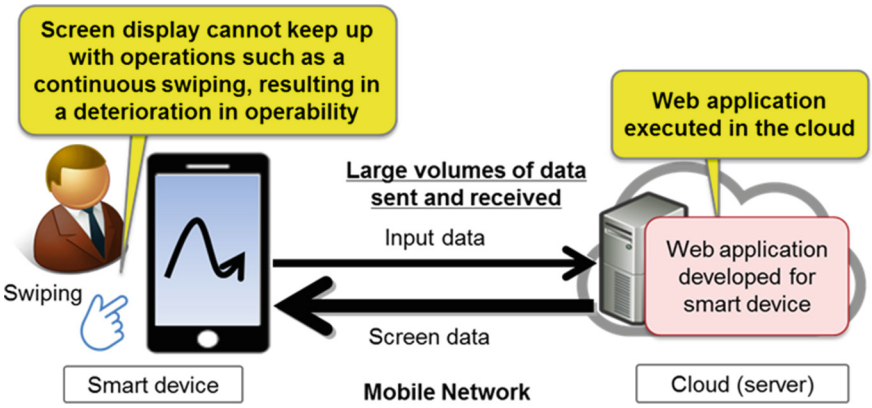

Fig. 2. Reduced responsiveness

We have a developed a virtualization technology for mobile web applications ("VMA") that automatically splits a web application into two parts—UI processing and data processing—in the cloud. VMA maintains security by processing and storing data remotely in the cloud; at the same time, it also makes applications highly responsive by handling UI interactions locally on each device. This technology allows service providers to offer existing line-of-business web applications that are highly responsive and secure at a low cost.

2 Issues and Existing Technology

VMA distributes and executes stand-alone web application for smart devices. This technology distributes a web application's logic so that smart devices handle UI interactions locally and servers in the cloud process data remotely. The UI and data processing parts of the application assume they always have a network connection

through which to communicate with each other. Because, the UI and data processing parts usually executed on same device. However, it's easy for smart devices, which often use mobile networks, to get disconnected frequently. If a web application is run with distributed UI and data processing in this kind of environment, dropped network connections can cause data loss, unexpected application behavior associated with failed requests, and reduced responsiveness associated with high network latency.

To address these issues, existing technologies have cached data in local storage for offline use [2] and also gracefully degraded to limited feature sets while offline [3].

Before an application can use internal storage or graceful degradation, however, every part of it that sends or receives data over the network needs to be updated accordingly. Furthermore, every new feature that uses a network connection would also need to deal with being offline. This additional processing has entailed increased costs.

3 Data Synchronization with VMA

In response to these issues, we developed a data synchronization technology with VMA that does its best to compensate for dropped network connections. This technology buffers network traffic and does not require developers to edit existing web applications' source code directly. Figure 3 illustrates the data synchronization technology in more detail while providing an overview of its architecture.

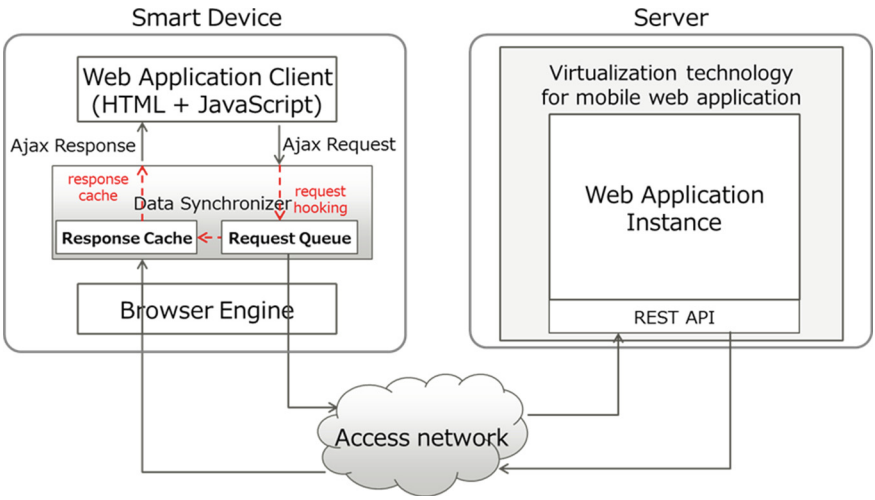

Fig. 3. Data synchronizer architecture

3.1 Overview

The data synchronization technology has a DataCache feature that hooks into the XMLHttpRequest API through which web applications send data to and receive data from the cloud. This feature allows requests to be sent from cached data even while

offline. The data synchronization technology also has a DataShrink feature that detects duplicate data to minimize network traffic. The data synchronization technology use transparent proxy technique for unmodified embedding into web application. These features reduce the amount of data that needs to be sent and received and make applications pleasantly responsive.

3.2 DataCache

The DataCache feature hooks into the XMLHttpRequest API to cache all upstream and downstream data.

The DataCache feature replaces the XMLHttpRequest browser API used by web applications with its own Hooking API. Because the Hooking API presents the same interface as the XMLHttpRequest API provided by browser engines, web applications do not need to be modified to use it. A transparent proxy adds the necessary code to load the Hooking API when a web application is transferred, thus replacing the XMLHttpRequest API and enabling the DataCache feature without burdening the web application developers.

The DataCache feature caches data that can largely be classified into two groups. The first group comprises requests that do not involve fetching data—specifically, GET (for posting parameters), SET, PUT, and other XMLHttpRequests. These requests are not considered to affect a web application over short periods of time, even if they are delayed in reaching the server. The second group comprises requests that involve fetching data (responses)—specifically, GET, POST, and other XMLHttpRequests. The DataCache feature handles these requests according to the following algorithm.

(A) While online, all requests (whether or not they involve fetching data) are immediately sent to the cloud; responses are sent to the web application and the response data is simultaneously cached.
(B) While offline, requests (whether or not they involve fetching data) cannot be sent to the cloud and are thus added to a queue. Cached response data is returned to requests that require a response.
(C) When a network connection is re-established, all queued requests are sent to the cloud in FIFO order. Cached response data is overwritten by any newer responses that are received.

By following this algorithm, the DataCache feature does its best to preserve web application functionality even while a client device is offline. Figure 4 illustrates how this algorithm works.

3.3 DataShrink

The DataShrink feature minimizes the amount of data that needs to be sent by checking both the content of requests when they are queued by the DataCache feature and the content of cached response data when it is sent or updated by the DataCache feature.

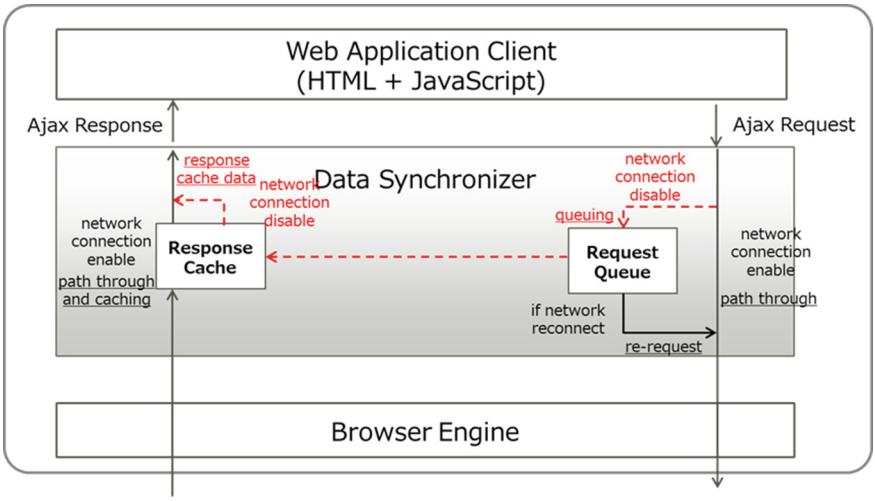

Fig. 4. Data synchronizer DataCache algorithm

The DataCache feature queues request data and pre-caches response data as a precaution against dropped connections. Though the DataCache feature queues and caches all request and response data, the uniqueness of that data does not necessarily need to be preserved. For example, a series of periodic requests to get all the latest data can be saved as a single request (along with its response) without any adverse effects on a web application's data integrity. On the other hand, periodic requests to get the latest data via a series of deltas and requests that contain data must all be run individually and their responses must be preserved. The DataCache feature sends its queued requests out to the cloud in FIFO order when a network connection is re-established, but as explained above the uniqueness of all that request and response data does not need to be preserved. Sending data without uniqueness constraints thus leads to unnecessary network traffic, wasted bandwidth, and slower response times.

By automatically identifying and eliminating superfluous requests and responses when unique copies do not need to be preserved, the DataShrink feature can avoid wasting network bandwidth and prevent decreased responsiveness. The DataShrink feature uses the following algorithm to determine which of a web application's requests and responses must remain unique.

(A) If a request has one or more mutable parameters, a unique copy of its data must be preserved.
(B) If a request does not have any mutable parameters, a unique copy of its data does not need to be preserved.

The following pseudocode shows how a request is checked for mutable parameters.

```
var history = [];
var dynamicRequests = [];
var staticRequests = [];

if (request == 'http') {
  if (request.hasOnTail(/?||&/)
    || !history.urlsuffixArray.has(request.urlsuffix)
    || (history.urlsuffixArray.has(request.urlsuffix)
        && !history.body.hasKey(request.body.key))) {
    dynamicRequests += request;
  } else {
    staticRequests += request;
  }
  history += request;
}
```

The DataShrink feature considers it necessary to preserve the uniqueness of a request when it has parameters; when it is the first request sent to a particular endpoint (URI); or when it contains the same data as some other past request (i.e. "A" above). Uniqueness is not considered a requirement for any other requests (i.e. "B" above).

The following pseudocode shows how the DataShrink feature removes redundant requests and responses that it has determined it does not need to uniquely preserve.

```
var mergeRequests = []
for (var i = 0; i < staticRequests.length; i++) {
  if (!mergeRequests.has(staticRequests[i].url) {
    mergeRequests += staticRequests[i];
  } else if (mergeRequests.has(staticRequests[i]) {
    var oldentry = mergeRequests.has(staticRequests[i]);
    if (staticRequests[i].requestDate >
oldentry.requestDate) {
    mergeRequests[staticRequests[i].key] =
staticRequests[i].value;
    }
}
```

This code extracts requests with unique URIs from the set of requests identified in step "B" of the algorithm above (staticRequests). If more than one request has the same URI, only the most recent one is selected.

Using these algorithms and procedures, the DataShrink feature minimizes the amount of data that is sent and received.

4 Results

To evaluate our data synchronization technology, we measured the following three metrics associated with web applications on a mobile data network: data retention rates, responsiveness, and bandwidth usage.

Specifically, we compared the performance of three different types of web applications with versions of those applications that dynamically loaded our data synchronization technology. The architecture of a web application that VMA has split into UI and data processing components is similar to the architecture of a web application that developers have split into server and client components; as a result, we can compare performance by applying (or not applying) our data synchronization technology to an entire web application.

We chose the following three web applications.

- A to-do application.
- A text chat application.
- A video chat application.

We arranged to test these three types of applications because they cover a spectrum of real-time demands on processing that needs to communicate with the cloud. The to-do application does not have real-time constraints because it can take a long time to update a to-do list on a server without any adverse effects. The text chat application has real-time constraints of several seconds or less because user input intervals are generally several seconds or more. The video chat application has real-time constraints of one second or less because on-screen video is updated by at least one frame per second.

We prepared a mobile network environment under simulated conditions in which devices would be randomly disconnected for 9 s at a time. We assumed that devices could spend 9 s offline before reconnecting because it takes devices up to about 9 s to reconnect after entering Sleep Mode and losing their network connection [4].

Figure 5 shows a comparison of the data retention rates measured when network connections were dropped for both unmodified web applications and web applications that loaded our data synchronization technology.

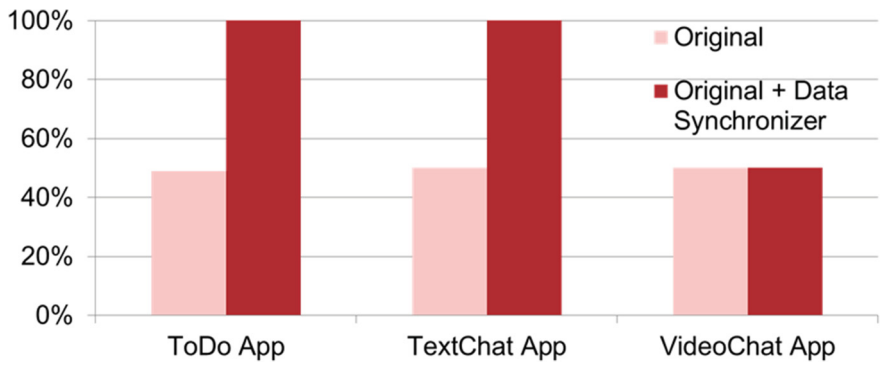

Fig. 5. Data retention rates (Color figure online)

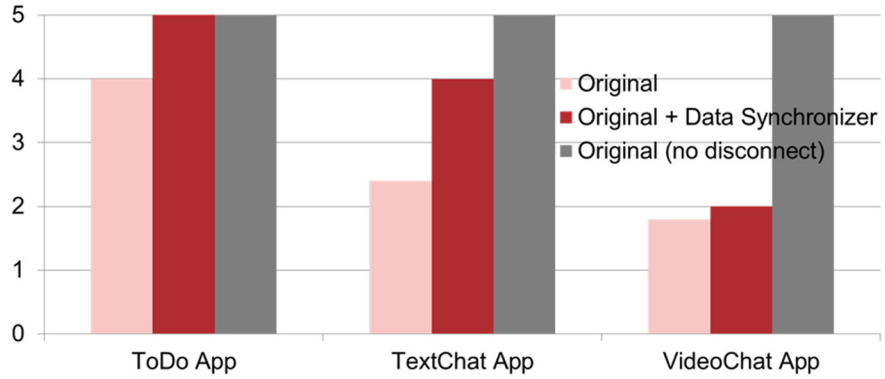

Fig. 6. Responsiveness survey results (Color figure online)

The results show that the to-do and text chat applications achieved 100 % deliverability rates without any data loss when they loaded our data synchronization technology, regardless of their connection patterns.

We also evaluated the web applications' responsiveness and bandwidth usage when they lost and subsequently re-established a network connection. To compare responsiveness, we had multiple test subjects try to use each of the three web applications with a randomly selected set of conditions from the following list. The test subjects then rated each application's ease of use on a 5-point scale.

- The original web application on a network connection that would be randomly dropped for 9 s at a time.
- The original web application with data synchronization on a network connection that would be randomly dropped for 9 s at a time.
- The original web application on a network connection that was never dropped.

To evaluate bandwidth usage when losing and re-establishing a network connection, we measured the amount of traffic generated by the three web applications both with and without dropped connections; we then made a relative comparison of the results.

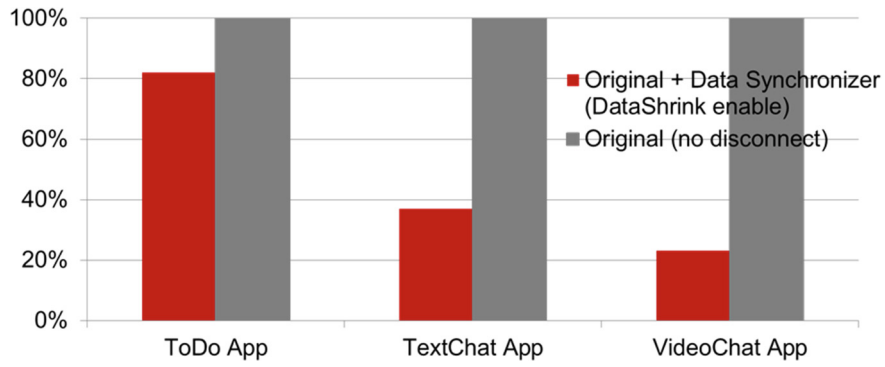

Fig. 7. Bandwidth usage (Color figure online)

Figure 6 shows a comparison of UI responsiveness and Fig. 7 shows a comparison of bandwidth usage.

The results show that our data synchronization technology was able to reduce the perceived loss of responsiveness accompanying dropped network connections for both the to-do application, which has no real-time constraints, and the text chat application, which has real-time constraints on the order of several seconds. On the other hand, the results also show that our data synchronization technology was not able to reduce the perceived loss of responsiveness for the video chat application.

Our evaluation of bandwidth usage revealed savings of approximately 20 % for the to-do application and 60 % for the text chat application when our data synchronization technology was loaded.

5 Conclusion, Discussion, and Future Work

The DataCache and DataShrink features of VMA's data synchronization technology buffer web applications' network communications to compensate for frequently dropped connections on mobile networks. Our experimental results show that web applications with real-time constraints of several seconds or less remain just as usable and responsive when they are temporarily offline. On the other hand, we also found that our data synchronization technology could not compensate for dropped network connections in web applications with real-time constraints of one second or less.

Furthermore, our DataShrink feature was able to reduce bandwidth usage and improve responsiveness by eliminating redundant web application requests and responses.

In addition to the data synchronization technology introduced in this paper, VMA also has the ability to dynamically adjust UI and data processing in accordance with user input. Together, these features allow VMA to make web applications safer and more responsive.

We plan to apply our data synchronization technology to an even wider range of web and native applications for further evaluation.

References

1. Fujitsu Laboratories Ltd.: Fujitsu Laboratories Develops Virtualization Technology that Brings Security and Operability to Web Applications, 29 May 2015. http://www.fujitsu.com/global/about/resources/news/press-releases/2015/0529-01.html
2. Ijtihadie, R.M., Chisaki, Y., Usagawa, T., Cahyo, H.B., Affandi, A.: Offline web application and quiz synchronization for e-learning activity for mobile browser. In: 2010 IEEE Region 10 Conference, TENCON 2010, 21–24 November 2010, pp. 2402–2405 (2010)
3. Goncalves, E., Leitao, A.M.: Offline execution in workflow-enabled Web applications. In: 6th International Conference on the Quality of Information and Communications Technology, QUATIC 2007, 12–14 September 2007, pp. 204–207 (2007)
4. Chen, W.-P., Licking, S., Ohno, T., Okuyama, S., Hamada, T.: Performance measurement, evaluation and analysis of Push-to-Talk in 3G networks. In: IEEE International Conference on Communications, ICC 2007, 24–28 June 2007, pp. 1893–1898 (2007)