Mobile Design Patterns: Push, Don't Pull

An IBM Redbooks Point-of-View publication by IBM WebSphere® Marketing

By **Jerry Cuomo**, IBM Fellow, VP, WebSphere Chief Technology Officer, and **Robert Vila**, Program Director and STSM, WebSphere Strategy and Emerging Technology

Highlights

- For mobile applications, a push design model can be vastly more efficient than a pull model.
- At peak query times, a push model can consume as little as 4% of the resources needed for a pull model, according to the authors' estimates.
- Mobile users often pull (query) data that has not changed, putting needless strain on server and network resources.
- The authors suggest pushing the most popular data to users' devices *as it changes*. Push notifications are much less resource-intensive than queries.
- In a mobile banking scenario, the authors envision someday removing the Check Balance button from apps because balances would *always* be up-to-date.

Push versus Pull: Is there a clear winner for mobile?

Designers of enterprise mobile apps have the opportunity and responsibility to think differently about application design, particularly when compared to their traditional web application counterparts. Few examples illustrate this better than the debate over *push* versus *pull*. Should we wait for users to request, or "pull", the content that they need, or should we "push" content updates when we think that users need them?

The answer naturally is "it depends". Different situations call for different solutions. But, current trends make us think that the push model has distinct advantages in the mobile space.

The familiar three-tier web design pattern, which is driven by user-initiated requests for content, does not always fit the mobile world. The sheer volume of mobile users and increasing ubiquity of mobile devices can lead to exponentially higher numbers of requests that can strain network and server resources to the limit.

The mobile push pattern, in contrast, sends content to devices automatically, eliminating the dependency on web app servers for pulling content updates. Instead of users continuously pulling or querying information on the odd chance that it has changed, updates are sent proactively using lightweight, integrated messaging services that can be scaled to meet the rapidly increasing mobile demand.

In short, in the proper situations, a push-based design can be an order of magnitude more efficient than a conventional pull design, while delivering a transformative experience for mobile users.

To be clear, we do not argue that web servers are unnecessary in mobile back-end services. In fact, we think that hybrid mobile apps that use HTML5 and JavaScript are excellent ways to provide services that run on a variety of devices.

However, it is important to think about the *right* architecture to take advantage of the capabilities of mobile devices, particularly when your choice can lead to breakthroughs in efficiency.

The push design pattern can be illustrated through a simple example scenario that, if extended to its logical conclusion, could make the Check Balance button in a mobile banking app a thing of the past. This IBM® Redbooks® Point-of-View article describes such a design and attempts to quantify its benefits. It also explores how data and usage analytics can be used to determine which model, push or pull, should be used in particular enterprise applications.

Scenario: Eliminating the need for the Check Balance button

Several large banks have told IBM that their “mobile apps are crushing IT” and that transactions with relatively low value to the bank are being frequently, almost whimsically, performed morning, noon, and night. One bank went so far as to say that a feature in its new mobile banking app that allows users to check their current account balance is overused and fits the whimsical, lower-value transaction profile.

To keep up with the high demand for account balance inquiries, the bank decided it needed to significantly increase the server and software capacity of its back-end systems. This would have included more application servers, data caching servers, security servers, and MIPS for its mainframe systems.

After examining the situation, IBM analysts suggested that the bank instead remove the Check Balance button from its mobile app. At first, the bank CIO looked at us as though we were crazy. But we explained that in place of the current query-based interaction design, we could try a mobile-oriented, push-based design instead. We estimated that a push-based design would be approximately 25 times more efficient than the query-based design.

This push design is simple yet powerful. The basic principle is that whenever a transaction occurs in a specific account, the new account balance is then pushed to the account holder’s mobile device by using a push notification from the source transaction system.

To illustrate the push design pattern, Greg Truty, one of the leading IBM mobile architects, designed a prototype of an iOS Bank Account Pass (see Figure 1) that resides in the iOS Passbook app, a popular tool used for hosting objects such as airline tickets and retail loyalty cards. As shown in the figure, the pass becomes a live view of the user’s bank account. It can even act as a handy substitute for the mobile banking app. This is because, along with the current account balance, Greg included details about the last posted transaction and added a QR code that can transform the pass into a debit card.

Of course, an iOS Bank Account Pass is just one example of how the bank can use push notifications to communicate balances and other information to customers. It can use the same pattern to add push notifications to its existing app so that users always have their latest balance and no longer need to keep pressing the Check Balance button. Regardless of how it is implemented, a push architecture can improve both IT efficiency *and* the customer experience.

Now that we’ve looked at the example, the next sections attempt to quantify the benefits of the push design pattern and examine how its use can be enhanced by using data and usage analytics.

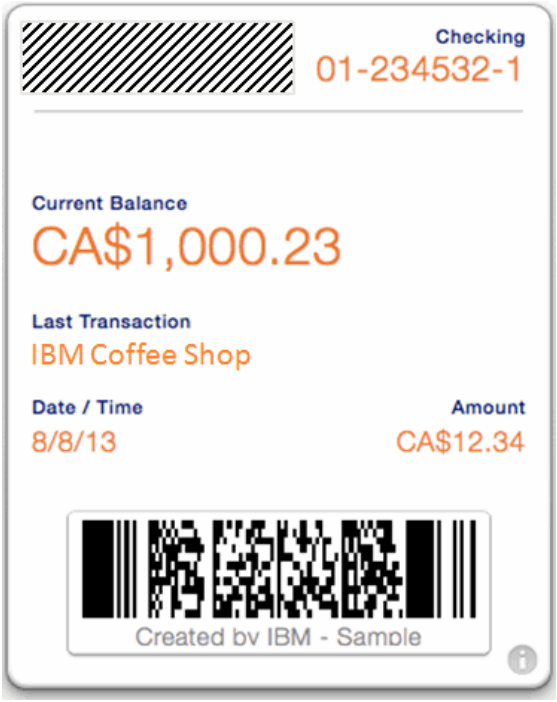

Figure 1 Prototype iOS Bank Account Pass

Pull pattern versus push pattern

In the classic, three-tier web design architecture (see Figure 2), users begin the interaction by loading a page in their browser. On the server side, requests are routed through a secure gateway and passed to the web server where static and dynamic content are returned. Back-end services, such as databases and other systems of record, are used to enrich content, personalize the results, and store session data.

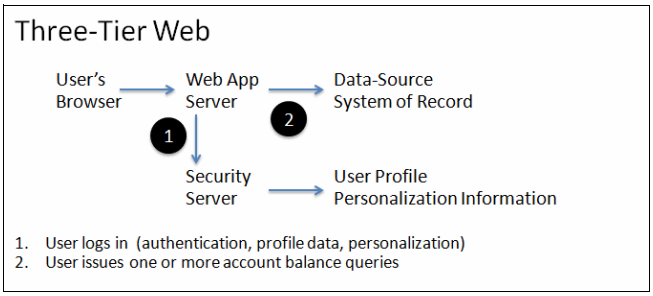

Figure 2 Traditional, three-tier web design architecture

The three-tier web design model is fundamentally built around the concept of pull. The process is driven by user-initiated requests for content. Web developers often use AJAX to make experiences more interactive, but the initial catalyst remains the user request.

However, to take advantage of the unique strengths of mobile devices, we can update our application architecture to support push-style communications.

In this pattern (see Figure 3), the user performs a one-time registration with their mobile app. From that point, data is pushed to the mobile device to keep account information current. The systems of record remain an important part of the architecture, but the dependency on the web app server for pulling account updates is eliminated. Instead, messaging services designed for reliable delivery and mobile scaling are used to deliver the content.

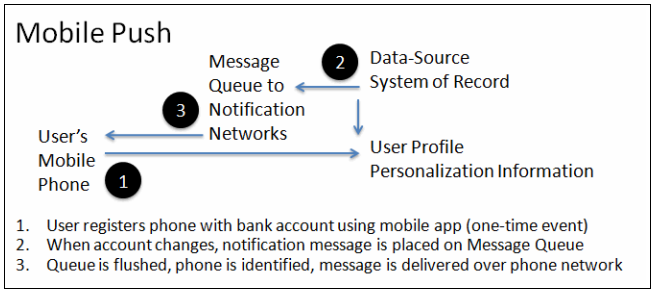

Figure 3 The push design model

Quantifying mobile design efficiency

Before we compare the push and pull patterns, consider how mobile has fundamentally changed when and how customers interact with banks and the resulting workloads.

In the past, banks hired extra tellers to handle the rush of customers at their branches during lunchtime and on paydays. With the advent of Internet banking, customers began to check their balances and perform other transactions from their desks at work or at home. Today, in the emerging era of mobile banking, customers can access their accounts anytime, from anywhere. If someone has just swiped their debit card for a latte, why should they wait for a statement at the end of the month when they can check their new balance immediately? The popularity of mobile banking has replaced some traditional branch and web traffic with a more uniform, but significantly increased, load on servers.

A simple, non-scientific example can illustrate the efficiency of a mobile push design versus a traditional web interaction design. The example (see Table 1) assumes that one million users interact with their balances each day and accounts for both peak and average loads. For simplicity, we use the arbitrary term *load units* to provide an order of magnitude estimate for the cost of operations, such as logging in and checking a balance. We have estimated the number of load units that are created for each typical customer interaction based on our experience with banking systems.

In the traditional pull web model, customers will log in and check their balance several times a day. We estimate that the average customer will generate 20 load units, for a daily total for all users of 20 million load units. However, the biggest challenge is the peak load during typical online banking periods. Our experience shows that 90% of server load can occur during this 4-hour window, leading to a peak of 4.5 million load units each hour.

How would a push-based workload model compare? In this model, customers no longer need to log in to learn whether their balance has changed. Their mobile app always has their latest balance and most recent transactions. By eliminating unnecessary logins and balance checks, we have cut a significant amount of the transaction volume. Assuming that the balance only changes twice a day, the total server load is now only 4 million load units a day. We have also eliminated the significant peaks in the pull model with more consistent, predictable workloads throughout each day.

Table 1 Comparison of push and pull models

Web workload (pull)	Mobile workload (push)
Basics ► Avg. user logs in 2x/day (2 x 5* = 10 load units) ► Checks balance 5x/day (5 x 2* = 10 load units) ► 90% of load occurs in two-hour windows each morning and afternoon **Usage** ► One million users ► 20 million load units/day **Distribution** • 9 million load units during each two-hour peak • 4.5 million load units each hour during peak periods	**Basics** ► Avg. user's balance changes 2x/day ► Two notifications/day (2 x 2* = 4 load units) ► Peak loads are spread evenly throughout the day **Usage** ► One million users ► 4 million load units/day **Distribution** ► Average of 0.17 million load units each hour, with no predicted peaks based on time of day
*Assumes five load units for each user login, two load units for each balance inquiry, and two load units for each push notification.	

The example shows that the push model is significantly more efficient than the traditional web model. By taking advantage of push capabilities, we reduced the daily server load by 80% and reduced peak levels to just 1/25 of their previous levels. Imagine the innovations that your organization can realize with this freedom.

Keeping the push just right by using analytics

Our example shows dramatic savings from switching from a pull to a push design. However, there are many cases where a push model is unlikely to provide significant benefits. In the example, the typical bank balance changed twice a day. But what if we look at a more active business account that might change hundreds or thousands of times a day? In this situation, constantly sending push updates might not be the ideal solution.

Fortunately, there is a way to get the best of both worlds. By using rules and analytics, we can determine the optimal way of serving the bank's different customers.

For example, after we provide both pull and push options, we can allow customers to select how often their accounts are updated. Many customers will choose to have immediate push alerts to keep their data current while others might prefer to have their data sent once a day. This approach allows customers to choose the option that best balances their needs for account accuracy, device battery life, data usage, and other factors. We can also use analytics to determine how often a particular customer checks an account, at what times of day, and whether there are any spikes in their usage. We can then proactively send account updates that are tailored to their schedule.

The flexibility of the push model also gives users the power to determine the device usage scenarios under which they want to receive updates. Some users might choose to receive account updates only when they are connected through WiFi or when they have plenty of available battery power. Other users might choose to skip push updates altogether or receive them only when they pass a specified low balance threshold or when a suspicious transaction is recorded. By supporting both transaction styles, a bank can deliver a transformative experience that is tailored to each user.

Summary

Our goal is to encourage clients to adopt design patterns that are built around the special capabilities of mobile. When the Internet was new, some companies simply replicated their existing "green screen" applications on the web. And because they did not grasp the transformational power provided by the new medium, many of those companies no longer exist today. We think the industry is at a similar inflection point with mobile apps. Replicating a web experience on a mobile device is not enough. The most successful companies will embrace the capabilities that are available on mobile and adopt architectures, such as the mobile push model, that can scale in this new world.

What's next: How IBM can help

We encourage you to adopt mobile design patterns, such as the push pattern demonstrated here, that take advantage of the unique capabilities of mobile platforms.

IBM offers comprehensive solutions for building, securing, and managing mobile applications to better serve your customers.

With the IBM MobileFirst portfolio, for example, your enterprise can deliver a truly transformative mobile experience. The MobileFirst portfolio includes analytics, testing, and customer experience tools that encompass all aspects of the mobile application lifecycle.

Resources for more information

For more information about the concepts highlighted in this IBM Redbooks Point-of-View article, see these resources:

- IBM MobileFirst

  http://www.ibm.com/mobilefirst/us/en/
- IBM MobileFirst Case Studies

  http://www.ibm.com/mobilefirst/us/en/see-it-in-action/
- IBM MobileFirst Solutions

  http://www.ibm.com/mobilefirst/us/en/offerings/

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

This document REDP-5072-00 was created or updated on December 5, 2013.

Send us your comments in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks