A Prototype Framework for High Performance Push Notifications

Emre Isikligil
Senior Backend Developer
Monitise MEA

Semih Samakay
Senior Automation Engineer
Monitise MEA

Deniz Kılınç, PhD
Manisa Celal Bayar University
Department of Software Engineering

ABSTRACT
Push notifications are the significant tools to increase engagement with your app and improve user retention rates. Push notifications notify users with new information about an application, so they provide valuable and relevant updates even when the application is not running. This paper describes a software prototype framework, which is designed to send millions of push notifications using GCM and APNS cloud services. 13 separate test scenarios designed to measure its performance are run, upon which final results are discussed and conclusions are drawn.

General Terms
Software Design and Architecture, Software Performance Test

Keywords
Push notification, performance, system design, messaging, producer/consumer.

1. INTRODUCTION
There has been significant growth and evolvement in internet and mobile technologies in the last couple of years which caused mobile applications to spread out into various business areas. According to Statista, the overall number of mobile phone users reached 4.61 billion in 2016 and it is expected to grow to 4.77 billion in 2017 [1]. With the widespread use of smartphones, mobile applications have become an important element of our daily lives. Google's report "Micro-moments" indicates that 87% of mobile phone users have their smartphones with them all the time. Furthermore, they check their phones and mobile applications 150 times per day with an average of 177 minutes usage.

The outstanding growth of the mobile application world led to the emergence of two mobile application metrics named app engagement and app retention, different but related metrics used to define app performance. Mobile applications should provide users with up-to-date and relevant data to increase user-app engagement and maximize user retention rates. Traditional web applications poll data source constantly to fetch new data. Considering power consumption, processing power and mobile data usage polling becomes extremely expensive for mobile devices. This is where push notifications comes on the scene. Using it, data source notifies mobile clients when new data arrive. Notifying users with new information, users are provided with valuable and relevant updates even when the application is not running. How push notifications are used mostly depends on the business preferences and standard approaches [3]. Some of the most common uses are automated banking alerts, marketing offers, breaking news, location-based messages, application centered task reminders, and order updates.

Since push notifications are a must, manufacturers of most widely used mobile devices have come up with cloud-based push service solutions. Today Google Cloud Messaging (GCM) or Firebase Cloud Messaging (FCM), Apple Push Notification Service (APNS) and Windows Push Notification System (WNS) are used for sending push notifications to Android, IOS and Windows devices, consecutively. FCM and WNS are out of scope of this paper.

Both GCM and APNS enable developers to send data from 3rd party servers to mobile applications. Although push notifications are useful for providing users with relevant data, they must not be relied upon for carrying sensitive data. GCM does not guarantee that messages are delivered to mobile clients in time [4]. Moreover, persistent TCP/IP push technique which is used by both GCM and APNS is stated to be unreliable [5].

Designing an architectural infrastructure to send millions of push notifications is a challenging task, which requires talented staff to develop expertise in architectural topics such as availability, scalability, performance, multi-processing/threading, caching, and queue management.

This paper will discuss the software prototype framework designed and developed by Monitise MEA, which develops mobile-based software applications in the domains of banking, e-commerce, payment, security and travel to many leading companies (İş Bank, Yapı Kredi, TEB, THY, First Gulf Bank, Qatar Islamic Bank) [6].

The structure of this paper is organized as follows;

In next section, general methods and approaches are presented. Section 3 gives information about the architectural design of the proposed system. Section 4 presents the performance test results. Finally, section 5 concludes the outcomes and provides information about our future work.

2. METHODS
The proposed system uses GCM and APNS cloud services to send mobile push notifications to devices running Android and IOS, consecutively. First, individual devices register themselves to these cloud services and get unique IDs. When new data arrives, the system sends them to GCM or APNS with these unique IDs. Then, messages are delivered to mobile devices by these cloud services. Following steps explain methodology for sending push notification to Android devices using GCM.

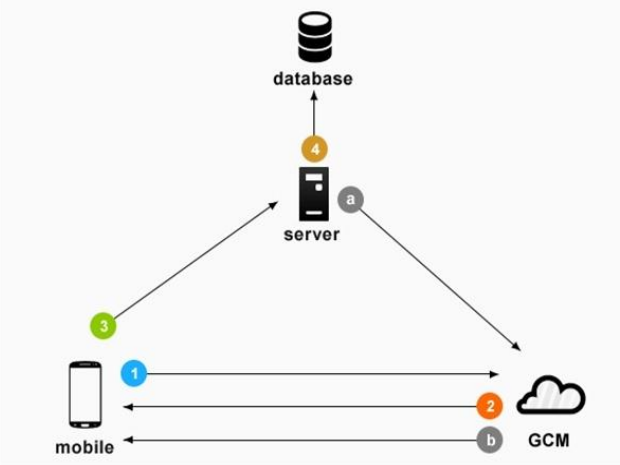

Figure 1: Android GCM architecture

1. A sender ID and app ID is sent to GCM by an Android device to register device.
2. If a successful registration is done, GCM sends a registration ID to Android device.
3. After receiving registration ID, the device sends the registration ID to server.
4. Server stores the registration ID in the database for later use.

5a. Server sends new data to GCM with registration ID.

5b. GCM delivers message to the mobile client.

They way iOS push notifications work is similar to Android push notifications. Required steps are given below.

1. iOS device requests a device token from APNS.
2. Device receives the token, which functions as the address to send a push notification to.
3. Device sends the token to application server.
4. Server stores the token for later use

5a. When prompted, server sends a push notification with the device token to APNS.

5b. APNS sends push notification to the mobile device.

3. PROPOSED SYSTEM DESIGN

The architecture of the prototype system consists of 6 modules; REST interface, service layer, database, message producer, message consumer and queue manager.

3.1 REST Interface

The REST interface is preferred because of its simplicity and ease of use in order to make the application communicate with the external world.

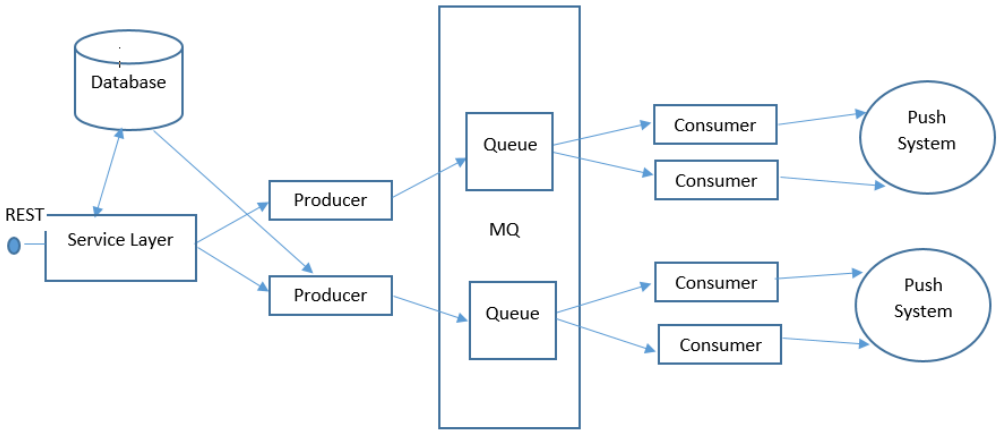

Figure 2: Architecture of Push Server

3.2 Service Layer

The service layer is the module that manages the application. It provides inter-module communication. The data provided from the interface is stored in the database by the services in this layer. This layer is also where the notification process is initiated to process the data in the database.

The scheduling service in the service layer can handle notifications scheduled to be sent later. In this way, it is also possible to process repetitive notification operations, such as forward notification procedures.

3.3 Database

The database is where the data needed by the application are stored. The data of mobile users, user groups, and information of notification processes are all stored in database. Database performance is directly related to push server performance.

3.4 Message Producers

The message producers are created by the service layer depending on the device type. They are used for preparation of notifications to be sent out and lining them up in the queues to be delivered to the consumers. During the dispatch of a notification process, a producer thread is created for each device type. These producers are responsible for generating notification messages by retrieving data from the related data tables.

Each producer is responsible for only one notification process. Threads are terminated after all messages have been created. If a new notification transaction is generated when another notification transaction is in progress, new message producers are created for the device types and current ones are not used for the new transaction.

3.5 Message Consumers

The message consumers receive messages waiting in the queue and send them to the notification systems depending on device type. Connection between the system and the cloud services is established by message consumers over HTTP. They can be positioned as threads in the main process, or they can operate as separate processes. In the scope of this paper, message consumers were created as threads. There are usually more than one consumer for a device type. Consumer numbers can be configured.

The queue manager has a different queue for each device type. Depending on the needs of the system, the number of consumers corresponding to these queues can vary and the transmission speed can be set within certain limits. The number of consumer varies dynamically depending on the instantaneous load of the application. Consumers do not know anything about of notification transactions they are processing. A consumer can send messages generated from multiple transactions. However, each consumer is created to send messages to only one notification system. Consumers do not have a specific running order. Any available consumer gets the message from the queue and continues with its process. Consumer threads keep running as long as there exist messages in the queue. When there is no message left to be consumed, message consumers are terminated by the application.

3.6 Queue Manager

The queue manager provides asynchronous operation of message producers and consumers. Incoming messages are queued up and served to consumers on a first come first served basis. Connection between the system and queue manager is established over JMS. With its own database, the queue manager can keep messages in the queue in case the system crashes.

4. EXPERIMENTAL RESULTS

Table 1: Configuration details of the environment

Application/ Component	Operating System	CPU Type	Clock Speed (GHz)	RAM (GB)	Software
Server	Ubuntu	Intel Xeon E5-2666 v3	2.9	3.75	Tomcat 8.0.33 JDK 1.8.0_91
Database Server	Ubuntu	Intel Xeon E5- 2670 v2	2.5	7.5	MySQL 5.6.23

Table 1 shows the hardware and software configuration details of performance test environment. In this study, 13 test scenarios were designed to measure the performance of the proposed system and each one was tested separately. Table 2 shows name, system version and load for each test scenario.

Table 2: Test scenarios

Test	Version	Load
T1	0.1.0.2	1m GCM
T2	0.1.0.2	1m APNS
T3	0.1.0.2	500k GCM + 500k APNS
T4	0.1.0.2	1m GCM
T5	0.1.0.24	1m GCM
T6	0.1.0.24	1m APNS
T7	0.1.0.24	1m GCM
T8	0.1.0.24	1m APNS
T9	0.1.0.24	1m GCM
T10	0.1.0.24	1m APNS
T11	0.1.0.24	500k GCM + 500k APNS
T12	0.1.0.24	500k GCM + 500k APNS
T13	0.1.0.24	500k GCM + 500k APNS

Non-physical iOS and Android devices were added to the database with random push IDs for one time only before running tests. Also physically known two devices (Nexus 5, Android 6.0 and iPhone 6s, iOS 9.2) were included in order to see actual push notification on device screen.

Push notifications were sent to 1 million devices in each test. In order to be able to compare performance test results, 3 million devices were added to the database in 2 different applications since total number of devices in database might adversely affect the performance of the application. 1 million Android devices and 1 million iOS devices were added to App1 and 500k Android devices and 500k iOS devices were added to App2. Tests containing only one of the platforms were run with App1 while tests containing both of them were run with App2. During the tests, asynchronous requests for saving device and user were made to the system to make scenarios more realistic. Table 3 shows test durations and throughputs for all tests. Concurrent threads is the number of consumers threads running concurrently. This parameter is configured separately for GCM and APNS. Tests were run for two different version of the application. In version 0.1.0.24, connection pools are employed for both HTTP and JMS connections.

Sending push messages to 1 million Android devices took 01:02:50 for the first test and the throughput of this test was 154 push messages per second (pps). Then, in the second test, 1 million push messages were sent to iOS devices and it took 01:25:33 with throughput of 195 pps. Number of consumer threads was 20 for both tests. Third test was done with mixed device types, 500k GCM and 500k APNS, where number of concurrent threads for each device type was 20. Application completed the task in 00:58:35 yielding throughput of 284 pps.

Before the fourth test, concurrent thread count was increased to 50 and 1 million push messages were sent to Android devices. The test was stopped because it has not finished after 2 hours and throughput was decreasing over time constantly. During the execution of the test, we figured out application run out of the memory. It can be said that T4 is an outlier since performance of the application could not be measured correctly because of the problem faced during the test.

Table 3: Test results

Test	Version	Concurrent Threads	Load	Duration	Throughput
T1	0.1.0.2	20	1m GCM	01:02:50	265
T2	0.1.0.2	20	1m APNS	01:25:33	195
T3	0.1.0.2	20 - 20	500k GCM, 500k APNS	00:58:35	284
T4	0.1.0.2	50	1m GCM	02:05:31	98
T5	0.1.0.24	50	1m GCM	00:33:18	501
T6	0.1.0.24	50	1m APNS	00:32:11	518
T7	0.1.0.24	100	1m GCM	00:36:13	460
T8	0.1.0.24	100	1m APNS	00:40:21	413
T9	0.1.0.24	20	1m GCM	00:29:26	566
T10	0.1.0.24	20	1m APNS	00:29:51	558
T11	0.1.0.24	20 - 20	500k GCM, 500k APNS	00:20:49	801
T12	0.1.0.24	50 - 50	500k GCM, 500k APNS	00:20.14	824
T13	0.1.0.24	100 - 100	500k GCM, 500k APNS	00:23:25	712

Memory problem was fixed, connection pools were added to the implementation. Then, the application was tested with different combinations of consumer thread count and device type after this point. Options for consumer thread count are 20, 50 and 100 where that of device type are GCM-only, APNS-only and mix of GCM and APNS

Throughput of the application in tests T5, T7 and T9 where push messages were only sent to GCM was 501, 460 and 566 pps, consecutively. In tests T6, T8 and T10 where push messages were only sent to APNS, throughput was 518, 413 and 558 pps. When tests were repeated with mixed device types throughput became 801, 824 and 712 pps in tests T11, T2 and T13.

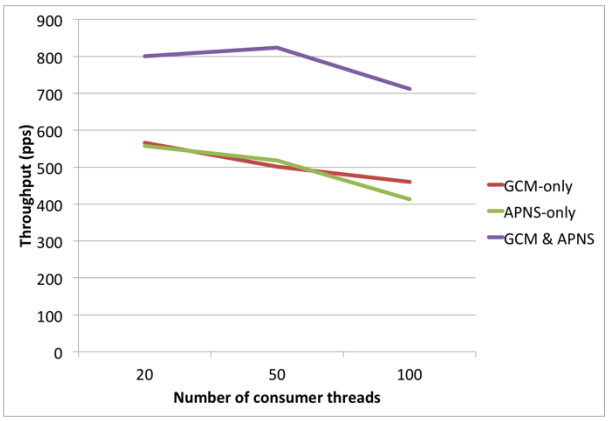

Figure 3: Throughput achieved with different number of consumer threads

Figure 3 shows how application performance is affected by number of consumer threads in terms of throughput. When messages were sent to only one device type performance decreased as the number of consumer threads exceeded 20. Similar behaviour were observed when push messages were sent to both device types. But, this time, performance started to decrease after the number of consumer threads exceeded 50.

5. CONCLUSION

Test results indicate that number of consumer threads have different influence on the performance of the application depending on the type of target devices. If target audience consist of people having mobile device of one platform only, number of consumer threads may be configured to be 20 for the best performance. On the other hand, if application is used for sending push notifications to both platform, consumer thread count may be set to 50. In order to achieve best performance the system can be configured accordingly depending on characteristics of the target audience.

Size of pools for HTTP and JMS connections was constant during performance tests. We anticipate that connection pool size would be another factor affecting application performance. In addition, processing power and hardware configuration would have an impact on performance. Similar tests can be conducted by adding these parameters to combinations of test scenarios as a future work. Furthermore, considering possibility of offering this system as a cloud solution, tests can be repeated by sending push notifications to more than one application at the same time.

6. REFERENCES

[1] Statista, “Number of mobile phone users worldwide from 2013 to 2019 (in billions)”, [Online], Available: https://www.appannie.com/insights/mwc-2016-top-stats-show-growth-mobile-market/

[2] Google, “Micro-Moments: Your Guide to Winning the Shift to Mobile”, [Online], Available: https://think.storage.googleapis.com/images/micromoments-guide-to-winning-shift-to-mobile-download.pdf

[3] Podnar, I., Manfred, H., and Mehdi, J. 2002. Mobile push: Delivering content to mobile users. In 22nd International Conference on. IEEE.

[4] Selim, Y., Aydin, B., and Demirbas, M. 2014. Google cloud messaging (GCM): An evaluation. In Global Communications Conference (GLOBECOM), 2014 IEEE.

[5] Li, N., Yanhui, D., and Guangxuan, C. 2013. Survey of cloud messaging push notification service. Information Science and Cloud Computing Companion (ISCC-C). In 2013 International Conference on. IEEE.

[6] Monitise MEA, “Monitise MEA web site”, [Online], Available: https://www.monitise.com/mea/mea