Hindawi Publishing Corporation
Mobile Information Systems
Volume 2015, Article ID 278713, 10 pages
http://dx.doi.org/10.1155/2015/278713

Research Article

Every Cloud Has a Push Data Lining: Incorporating Cloud Services in a Context-Aware Application

Tor-Morten Grønli,[1] Jarle Hansen,[2] Gheorghita Ghinea,[1,3] and Muhammad Younas[4]

[1] *Westerdals Oslo School of Arts, Communication and Technology, Faculty of Technology, Oslo, Norway*
[2] *Department of Computer Science and Information, Brunel University London, Uxbridge, UK*
[3] *Brunel University, Uxbridge, UK*
[4] *Oxford Brookes University, Oxford, UK*

Correspondence should be addressed to Gheorghita Ghinea; george.ghinea@brunel.ac.uk

Received 13 June 2013; Accepted 22 October 2013

Academic Editor: David Taniar

We investigated context-awareness by utilising multiple sources of context in a mobile device setting. In our experiment we developed a system consisting of a mobile client, running on the Android platform, integrated with a cloud-based service. These components were integrated using push messaging technology. One of the key features was the automatic adaptation of smartphones in accordance with implicit user needs. The novelty of our approach consists in the use of multiple sources of context input to the system, which included the use of calendar data and web based user configuration tool, as well as that of an external, cloud-based, configuration file storing user interface preferences which, pushed at log-on time irrespective of access device, frees the user from having to manually configure its interface. The system was evaluated via two rounds of user evaluations ($n = 50$ users), the feedback of which was generally positive and demonstrated the viability of using cloud-based services to provide an enhanced context-aware user experience.

1. Introduction

The design and development of context-aware solutions [1–4] aim to align information and services with the needs of users as well as the characteristics of mobile devices and underlying networks. The significance of context-aware solutions can be realised through a wide range of applications in different areas such as personalization of services, marketing, advertisement, healthcare, traffic management, and dynamic configuration of computing devices and networks. While a substantial amount of research has already been conducted on context-awareness [2, 5], the majority of existing techniques do not take account of the multisource context information in enhancing user experience and in remotely controlling mobile devices. In this paper we extend our previous work [6], which combines context-aware information from several dimensions, in order to remotely configure mobile devices such that they constantly adapt to the environment and change in accordance with the user's implicit requirements.

Specifically, we exploit cloud computing technology for creating new user experience and for controlling mobile devices. Previous research on cloud computing has focused on issues such as security [7–9] and performance [10–12]. However, cloud-based service providers and developers are increasingly looking towards the mobile domain, having their expectations focused on the access and consumption of services from mobile devices [13]. Hence, integrating an application, running on a mobile device, with cloud computing services is becoming an increasingly important factor. Moreover, high bandwidth wireless networks have become ubiquitous and are being used to connect mobile devices to the cloud. Potentially, by utilising such connectivity to offload computation to the cloud, we could greatly amplify mobile application performance at minimal cost [14].

Large IT companies like Microsoft, Google, and IBM all have initiatives relating to cloud computing which have spawned a number of emerging research themes, among which we mention *cloud system design* [8, 15], *benchmarking*

of the cloud [16], and *provider response time comparisons*. In addition, Mei et al. [17] have pointed out four main research areas in cloud computing that they find particularly interesting, namely, the *Pluggable computing entities, data access transparency, adaptive behaviour of cloud applications, and automatic discovery of application quality.*

In this paper we focus on *data access transparency* (where clients transparently will push/pull data to/from the cloud), *adaptive behaviour of cloud applications*, and *cloud benchmarking*. Accordingly, we adapted the behaviour of the Google App Engine server application based on context information sent from the users' devices, thus integrating context and cloud on a secure Android platform. Moreover, in doing so, we also examined the performance of cloud-based messaging services.

Contributions of this work are summarized as follows:

(i) Development of new techniques exploiting cloud computing technology for controlling mobile devices.

(ii) Data access transparency, ensuring that clients can transparently push/pull data to/from the cloud, thus realizing the adaptive behaviour of cloud applications.

(iii) Cloud benchmarking and performance evaluation of cloud-based messaging services, such as push messaging for the Android platform.

All of the above contribute towards a new user context-aware experience, as demonstrated through a proof-of-concept prototype. Accordingly, the paper is structured as follows. Section 2 describes experiments looking at performance issues when pushing messages from the cloud to various mobile devices. Section 3 then details how one can take advantage of cloud-based push messaging to provide tailored context-aware services, through development of a novel prototypical Android application. Thereafter, Section 4 presents evaluation results of the prototype, while Section 5 discusses findings and their implications. Finally, Section 6 concludes our paper.

2. Cloud to Device Messaging: Investigation of Push-Based Solutions

One of the main technical goals with the prototype created for the study described here was to make the interaction between the cloud and mobile devices as seamless as possible for the user. Moreover, in this paper, we focus on context management in a cloud environment. Specifically, we wanted to provide information that was relevant to the current situation, and this meant combining several different sources of context. Context management itself in our approach is realised through a cloud-based server, which communicates with smartphone clients via push messaging. However, there are several alternative methods to implement push messaging for the Android platform, and that is why we firstly wanted to do an in-depth investigation to find the best-suited technology for our main experiment. Our efforts in this respect are presented next.

2.1. Cloud to Device Messaging. *Cloud computing* and *mobile applications and media tablets* are both on Gartner's [18] top 10 list of strategic technologies for 2013, highlighting the current importance of these technologies. However, in order to provide the integration between smartphones and cloud computing services, there is a need for network communication.

To achieve this, there is the need for data-delivery mechanisms, of which three main categories can be distinguished [19]: (1) *Pull-based* (on-demand), (2) *Push-based* (publish-subscribe), and (3) *Hybrid*. The pull-based messaging model is where the user device or computing system pulls data from the service provider's system. According to [19], the advantage of this model is that no unsolicited or irrelevant data is received at the device. This is because it uses an on-demand model, where the data is received only when the device asks for it. The disadvantages of this model include the difficulty of setting the correct frequency of requests. Setting the frequency too fast drains the battery and uses a considerable amount of bandwidth. However, configuring with a slower frequency can cause the data to become old. Another disadvantage is the amount of server resources used. When many mobile devices are constantly polling for updates, it can cause a considerable load on the system.

Due to the mentioned disadvantages with a poll-based solution, we opted to use a push-based data-delivery mechanism. In doing so, we also focused on the integration of the cloud-based server, which published information to the registered smartphone clients.

2.2. Push Messaging Research. The push-based messaging model is where the server pushes data from the computing system to a client. It is also referred to as publish-subscribe, where the data is pushed to clients that have subscribed to a service, and is an asynchronous messaging model. Only data from subscribed sources will be received, which makes this paradigm loosely coupled [20]. An advantage of the push-based model is that there is no need to set the polling-frequency. The server will publish new data when it arrives. Kamal [19] states that one disadvantage with this model is the possibility of receiving unsolicited, irrelevant, or out-of-context data. While we acknowledge this possibility, the case remains that one will still have to subscribe to a service to receive information from it. Accordingly, in this section, we present a few relevant research efforts on the topic of push messaging.

There are several research efforts on the topic of push messaging and the integration of cloud computing with mobile applications. One example is the research effort described in [13], which describes *Bakabs*, an application created for Android and iOS, which relies on the cloud to bring its functionality on the provisioning of load management services in cloud-based web applications. Another paper utilising push messaging is detailed in [21], which focuses on the interoperability issues of cloud computing. Here, a generic middleware framework called *Mobile Cloud Middleware* (MCM) is created, which simplifies the use of process-intensive services from mobile phones. MCM uses an asynchronous notification feature to decouple the client and

server. This is used to address cases where the mobile client is expected to perform a time consuming task.

In our investigation, we provided a benchmarking test of push messaging technologies on the Android platform. Cloud computing benchmarking tests are available in previous efforts, with research such as that described in [10, 22], but in our case we tested the integration between cloud computing and smartphones. We also provide a different perspective in that we are specifically looking at the benefits and drawbacks of push messaging on the Android platform, which we have not seen in any previous research efforts.

2.3. Benchmarking Test. We conducted a benchmarking test to compare the relevant push messaging technologies for Android. We considered the six main push messaging libraries: *C2DM* (Cloud to Device Messaging), *Urban Airship*, *Xtify*, *XMPP* (Extensible Messaging and Presence Protocol), *MQTT* (Message Queue Telemetry Transport), and *Deacon*. We believe these technologies represent the most promising and useful push messaging libraries for Android; however, other interesting options may well be available. Of these, we chose not to include two specific libraries in our test, namely, MQTT and Deacon. MQTT was not included because we wanted to investigate push messaging technologies that can be easily integrated into the cloud, and specifically on the Google App Engine. MQTT is useful for connections that require a small code footprint and where network bandwidth is limited (http://mqtt.org/). It does require a message broker hosted on a separate server, and we did not find an easy way to integrate this service with the App Engine.

The second technology not included was The Deacon Project (http://deacon.daverea.com/about/). It is an open source project providing push notifications to Java and Android applications. This project was the least mature technology of the options we considered, as it is currently in the beta stage. Additionally, the project was created for users wanting to run push notifications on their own server and support Android versions lower than 2.2 (http://deacon.daverea.com/2010/04/welcome-to-the-deacon-project/), whereas C2DM requires at least Android 2.2. These requirements did not match what we wanted to investigate in this experiment, which included a close integration with a cloud-based server application and devices running on at least the 2.2 version of Android. Therefore, we included the following four technologies in our benchmarking test.

2.3.1. XMPP. XMPP is created for real-time communication (http://xmpp.org/about-xmpp/) and for streaming XML [20]. The technology behind XMPP was created in 1998 and then refined by the Jabber open source community in 1999 and 2000, before it was formalized by the IETF (The Internet Engineering Task Force) in 2002 and 2003. It is commonly used in Instant Messaging (IM) and is used by Google Talk, Jabber, and other IM networks.

2.3.2. C2DM. The next technology selected was C2DM. The overall goal of this library was to make it easier for mobile applications to sync data with servers (http://code.google.com/android/c2dm/). The message limit is set to 1024 bytes and developers are encouraged to send short messages, essentially notifying the mobile application that updated information can be retrieved from the server. Google has recently replaced this technology with the final version of the product, now renamed Google Cloud Messaging (http://developer.android.com/google/gcm/index.html).

2.3.3. Urban Airship. Urban Airship is a commercial option with support for Android, Blackberry, and the iOS platform (http://urbanairship.com/products/push-notifications/). In addition to offering C2DM support, it also provides a proprietary alternative. Urban Airship includes a push messaging alternative called *Helium*, which supports Android 1.6 and onwards.

2.3.4. Xtify. Similar to Urban Airship, Xtify is also a commercial option supporting multiple platforms (Android, Blackberry, and iOS). It also supports C2DM, but in addition it provides a proprietary protocol built on top of XMPP, using their internal infrastructure (http://developer.xtify.com/display/sdk/Xtify+Android+XMPP+Rich+Notification+Guide).

It is important to note that, even though two technologies in our benchmarking test are based on XMPP, we use XMPP directly when integrating with the Google App Engine. When testing Xtify we use their API and infrastructure, which provides a distinct difference between the two alternatives.

2.4. A Benchmark Test for Push-Based Messaging. For running the benchmarking test, we created a client application running on the mobile device. This client application was responsible for calling all the different push messaging technologies in sequence and then recording the time used. On the server side we implemented a Google App Engine application that sends messages when requested to do so from the client; it is also responsible for storing all the result data received from the mobile application.

Accordingly, the performance test followed these main steps:

(1) The Android client registers with the server. This is done differently for each technology, for example, C2DM will send a registration ID to the device.

(2) A timer is started on the client, followed by a message being sent to the server requesting a new push message.

(3) The server application receives the message and immediately sends out a push message consisting of 450 bytes to the mobile device. This will happen for each technology type.

(4) When the message is received, the Android client stops the timer and registers the result. This result is then sent back to a result-servlet that is part of the server application, which permanently stores the information.

(5) Finally, the process waits 5 minutes before continuing with the next technology.

As part of the benchmarking test, we used three Android devices, namely, *HTC Nexus One*, *HTC Evo 4G*, and *Samsung Galaxy Tab 10.1*.

Table 1: Benchmarking test results.

	Tech	Number of messages	Average response time (ms)	Standard deviation
Device				
Samsung Galaxy Tablet 10.1	C2DM	281	466.82	203.76
Android 3.1	Urban Airship	279	619.43	708.72
WIFI	XMPP	280	343.31	172.91
HTC Evo	C2DM	174	401.89	95.40
Android 2.3	Urban Airship	172	473.88	321.97
WIFI	XMPP	168	316.90	67.84
HTC Nexus One	C2DM	17	502.47	59.68
Android 2.3	Urban Airship	37	814.27	943.24
3G	XMPP	30	436.60	286.10
HTC Evo				
Android 2.3	Xtify	213	432.92	250.09
WIFI				
Message size				
	C2DM	281	535.20	281.03
Small (1 byte)	Urban Airship	277	407.44	3222.67
	XMPP	279	900.46	224.91
	C2DM	281	466.82	203.76
Medium (450 bytes)	Urban Airship	279	619.43	708.72
	XMPP	280	343.31	172.91
	C2DM	281	472.24	280.77
Large (900 bytes)	Urban Airship	279	356.37	3222.68
	XMPP	280	499.46	225.03

2.5. Benchmark Test Results. The goal of the benchmarking test was to compare push messaging technologies on the Android platform, and find the overall best-suited alternative for our experiment. We define performance in this context as follows:

(i) Response times: what are the response times for the different push messaging technologies?

(ii) Stability: are the response times providing stable results over the time we run the test?

(iii) Energy consumption: which technology provides the best energy efficiency?

Table 1 presents the overall results from the test. Using the results from the Samsung Galaxy tablet, the average response time for XMPP was the shortest, with C2DM being second and finally Urban Airship. There is a difference of 276.12 ms between the fastest (XMPP) and slowest (Urban Airship) average response times.

In Figure 1 we present the results gathered for the *Samsung Galaxy Tab 10.1*, and in Figure 2 are the results from the *HTC Evo*. Both devices ran on the same WIFI network and we were able to provide a fairly equal number of messages for each technology. As can be seen from the graph, the difference is mostly due to spikes in the response from Urban Airship. These spikes were more frequent in the results gathered from the Samsung Galaxy; however, they were also present during the other tests (e.g., with the HTC Evo device), which can also be confirmed when looking at the standard deviation. The maximum time used for Urban Airship was 5337 ms (*Samsung Galaxy*) and 3601 ms (*HTC Evo*), whereas both XMPP and C2DM provided considerably more stable

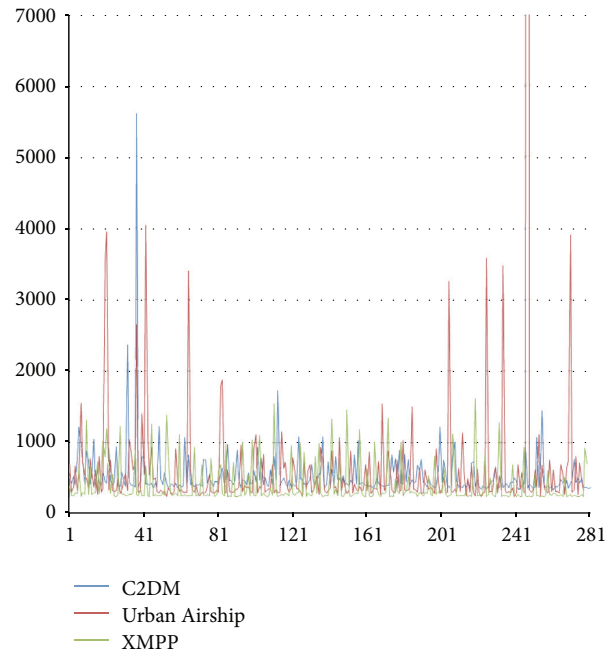

Figure 1: Results for Samsung Galaxy Tab 10.1 (WIFI)—small messages (1 byte).

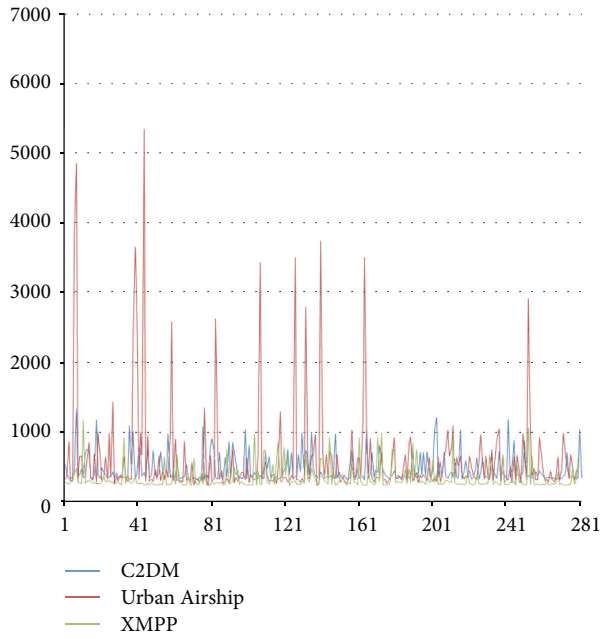

Figure 2: Results for Samsung Galaxy Tab 10.1 (WIFI)—medium messages (450 bytes).

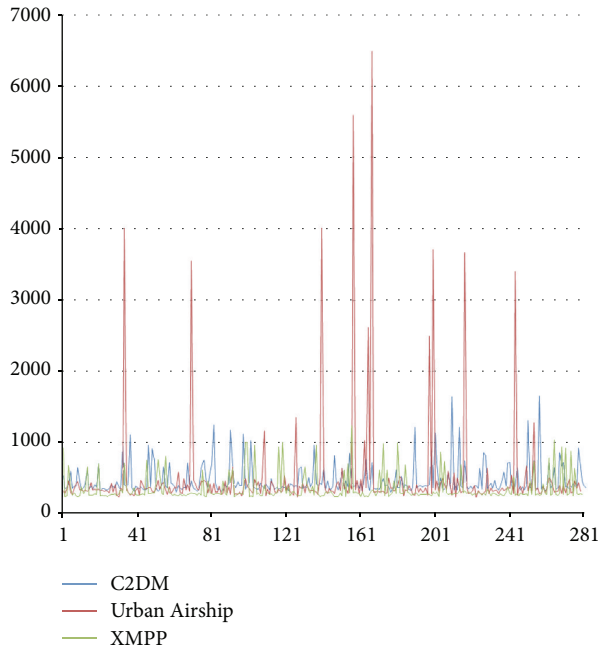

Figure 3: Results for Samsung Galaxy Tab 10.1 (WIFI)—large messages (900 bytes).

results. XMPP had the most stable results in our test, with a standard deviation of 172.91 ms (*Samsung Galaxy Tab 10.1*) and 67.84 ms (*HTC Evo*). Urban Airship did appear to have more stability issues than the rest and these issues surfaced several times during the test. When comparing the results from different WIFI networks and 3G, the same pattern emerges. The 3G response times are higher, but this is to be expected since they will have less bandwidth than the WIFI connection.

Overall the C2DM results were stable and the performance results recorded showed an average response time of 466.82 ms on the Samsung Galaxy tablet over 281 messages. With the HTC Evo we recorded an average response time of 401.89 ms over 174 messages.

The final technology we tested was Xtify, and it comes very close to the overall performance of C2DM with an average response time of 432.92 ms on the HTC Evo (again, running on a WIFI network). Additionally, it also provides more stable results than Urban Airship. We were unable to record more messages with Xtify, because of limitations with our developer account, which is also stated in the *limitations* section.

For the final test, we wanted to investigate the energy consumption [23] of the different push messaging technologies. In this performance test, we ran the same application as before, but we increased the time between messages to 10 minutes. Another difference was that, instead of running each technology in sequence, we only recorded one technology at a time. This was done because we wanted to provide results based on the battery level for each technology and also to expand the test over longer periods of time. The client still ran the test by sending requests to the cloud-based server, but we added a feature that triggered a new message for each change in battery level. By doing this, we were able to record the messages and also the corresponding battery level for the device.

The test ran on the *Samsung Galaxy Tab 10.1*. This was done to get a more comprehensive test, as the Samsung Galaxy Tab 10.1 has a significantly larger battery (7000 mAh) compared to, for instance, the HTC Evo (1500 mAh). In this test we included C2DM, Urban Airship, and XMPP.

With both C2DM and Urban Airship we were able to provide a fairly equal number of messages, with 862 and 858 messages sent, respectively. However, with XMPP, we were unable to send more than 295 messages because of quotas and limits in the Google App Engine (https://developers.google.com/appengine/docs/java/xmpp/overview#Quotas_and_Limits). The results are presented in Figure 3, and we have added trend lines for each result to make it easier to see the differences between the technologies.

Both C2DM and Urban Airship provided the best results, using less energy than XMPP. This was an expected result, and as stated by Xtify (http://developer.xtify.com/display/sdk/Xtify+Android+XMPP+Rich+Notification+Guide), the usage of XMPP is recommended in cases where one needs to communicate frequently over a short period of time. Both Urban Airship and C2DM provided fairly similar results, although, from the last 50% and towards 0%, Urban Airship did provide slightly better results. We are now in a position to employ cloud-based push messaging to smartphones in a proof-of-concept application, which we next describe.

3. Android Home Screen Prototype

We further developed a prototype of a customised Android home screen application, which consisted of three main

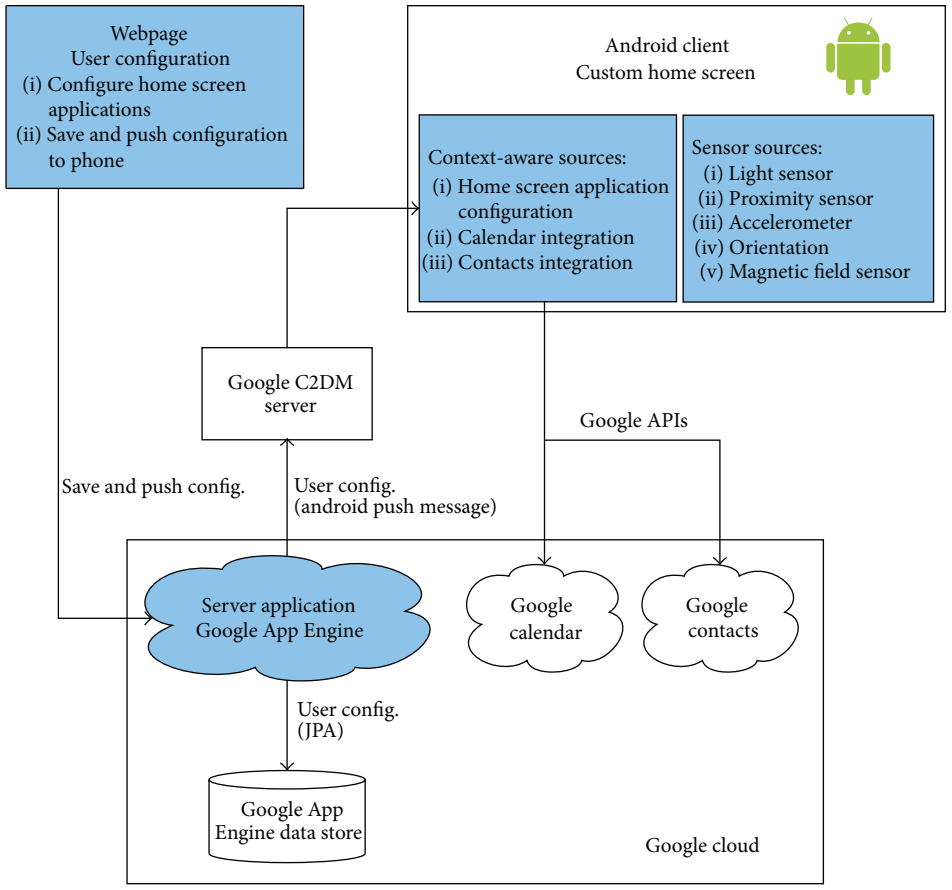

Figure 4: Application suite architecture.

components: (1) *Android client*, (2) *external Google services,* and (3) *a cloud-based server*. This prototype explored in practice the results obtained from the benchmarking test.

The Android client was the main component. It communicated with both the cloud-based server, installed on the Google App Engine, and the external Google services. The Google services provided data for calendar and contacts information, where the client directly communicated with them through an API provided by Google. The proof-of-concept created for the main experiment provided users with the ability to configure their Android device through the use of a cloud-based service. The mobile device collected context data from several sources and cooperated with the cloud server to provide useful features, such as a context-aware calendar and contacts list. The features included the ability to filter contacts based on the upcoming meeting and by meta-tags applied to the Google entities. A webpage contained all the configuration options available to the users, and thereby controlled the home screen applications on the smartphones. Push messages were sent to devices in the background, and the devices would be updated to reflect the configuration selected on the webpage.

As a part of our solution we chose the cloud computing platform in order to have a feature rich, scalable, and service oriented server framework. Traditional REST framework services were considered but found to be insufficient in terms of scalability and extensibility, that is, to add and remove context-aware sources in an ad hoc manner. The cloud-based approach also has the advantage of being run as a platform as a service instance in the separate hosting instance of Google App Engine.

For our user experiment we implemented an application suite, a fully functional demonstration of the system. One of the main technical goals of our system is to make the interaction between the cloud and the mobile device as seamless as possible for the user. The system was designed with three major components: an Android client, a cloud server application, and the remote Google services. Figure 4 gives an overview of the implementation of the system. The blue (or shaded) boxes in the diagram represent the parts of the system we created. The white boxes, like Google calendar and contacts, are external systems the system communicates with. The server application was deployed remotely in the cloud on the Google App Engine, whilst data was also stored remotely in Google cloud services.

After the Android client was installed on the mobile device, the device will register itself to Google. The users would start by logging in to the webpage. This webpage is part of the server application hosted on the Google App Engine. The login process uses the Google username/password. By leveraging the possibilities with Open Authorization (OAuth) the system provides the user with facility of sharing their

private calendar appointments and contacts stored in their Google cloud account without having to locally store their credentials. OAuth allowed us to use tokens as means of authentication and enabled the system to act as a third party granted access by the user.

After a successful authentication the user is presented with a webpage showing all configuration options. Because the configuration for each user is stored in the cloud, the system avoided tying it directly to a mobile device. One of the major benefits of this feature is that the user did not need to manually update each device; users have a "master configuration" stored externally that can be directly pushed to their phone or tablet. It is also easier to add more advanced configuration options when the user can take advantage of the bigger screen, mouse, and keyboard on a desktop/laptop PC for entering configuration values than those found on mobile devices.

The system exploits the push feature of Android 2.2 in order to send messages from cloud to devices, that is, the C2DM (Cloud to Device Messaging). The C2DM feature requires the Android clients to query a registration server to get an ID that represents the device. This ID is then sent to our server application and stored in the Google App Engine data store. When a message needs to be sent, the "save configuration" button is pushed. We composed the message according to the C2DM format and sent it with the registration ID as the recipient. These messages are then received by the Google C2DM servers and finally transferred to the correct mobile device.

The C2DM process is visualized in Figure 5. This technology has some very appealing benefits: messages can be received by the device even if the application is not running; it saves battery life by avoiding a custom polling mechanism; and it takes advantage of the Google authentication process to provide security.

Our experience with C2DM was mixed. It is a great feature when you get it to work, but the API is not very developer friendly. This will most likely change in the future since the product is currently in an experimental state, but it requires the developer to work with details like device registration and registration ID synchronization. Although C2DM does not provide any guarantees when it comes to the delivery or order of messages, we found the performance to be quite good in most of the cases. It is worth mentioning that we did see some very high spikes in response time for a few requests, but in the majority of cases the clients received the responses within about half a second. Performance measurements (which we recorded while doing the user experiments) reported an average response value of 663 milliseconds. It is also important to note that issues like network latency will affect the performance results.

4. Results from Prototype Evaluation

The developed prototype was evaluated in two phases. In the first, a pilot test was performed with a total of 12 users. These users were of mixed age, gender, and computer expertise. The results from this phase were fed back into the development loop, as well as helping remove some unclear

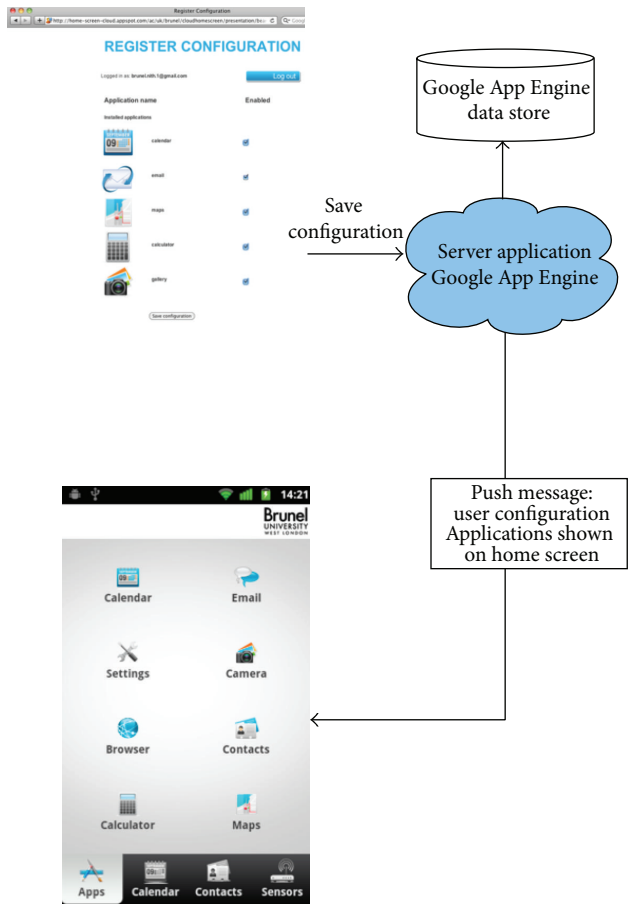

Figure 5: C2DM lifecycle.

questions in the questionnaire. In the second phase, the main evaluation, another 40 people participated. Out of the 40 participants in the main evaluation, two did not complete the questionnaire afterwards and were therefore removed making the total number of participants 38 in the main evaluation. All 50 participants were aged between 20 and 55 years and had previous knowledge of mobile phones and mobile communication, but had not previously used the type of application employed in our experiment. None of the pilot test users participated in the main evaluation.

In the main evaluation the users were given a mobile device running the application and were asked to complete a set of tasks. These tasks are presented in the following list.

User Tasks

(1) Start Android application; you might need to enter username/password and give the application access to the Google account the first time.

(2) Select calendar tab, and pay attention to the type of the first appointment.

(3) Select contacts tab. Make sure the contacts are filtered based on the type tag on the next meeting. If work is the type of the next meeting, only work related contacts should be shown.

Table 2: User evaluation questionnaire and results.

Statement	Domain	Mean	Std. Dev.
	Web application		
S1	I was able to register my device application configuration in the web application	3.61	0.59
S2	I was not able to store and push my configuration to my mobile device from the web page	1.47	0.80
S3	We would like to configure my phone from a cloud service on a daily basis, (webpage user config. and Google services like mail/calendar/contacts)	3.18	0.69
	Context-awareness		
S4	The close integration with Google services is an inconvenience. We are not able to use the system without changing my existing or creating a new e-mail account at Google	1.76	0.88
S5	Calendar appointments displayed matched my current user context	3.58	0.55
S6	The contacts displayed did not match my current user context	1.29	0.52
S7	I would like to see integration with other online services such as online editing tools (e.g., Google Docs) and user messaging applications (like Twitter and Google Buzz)	3.29	0.73
	Cloud computing		
S8	I do not mind Cloud server downtime	2.08	0.78
S9	I do not like sharing my personal information (like my name and e-mail address) to a service that stores the information in the cloud	2.16	0.79
S10	Storing data in Google Cloud and combining this with personal information on the device is a useful feature	3.26	0.60
S11	I find the cloud-to-device application useful	3.53	0.51

(4) Move to the sensor tab on the Android device. Select the different sensor options and look at how the values change when you move/shake/expose to light, and so forth.

(5) On your computer start a browser and perform the following steps on the designated website:

  (i) Select the application you want displayed on the home screen on the Android device.

  (ii) Press the "save configuration" button. The configuration is sent as a push message directly to your phone when this button is pressed.

(6) Make sure the home screen on the Android device (Apps tab) is updated and the icons you selected on the webpage are shown.

(7) Press one of the icons to launch the application and get back into the application afterwards.

(8) On the apps screen of the Android client, try to expose the phone to more/less light.

(9) Make sure the background changes colour based on the different light levels.

(10) Still on the apps screen, pick up the phone and shake it.

(11) The UI should now change to a simple layout. This is meant for users on the move (e.g., running/walking) where they might want to use a simpler and more concise layout. After 15 seconds of keeping the device still, the usual layout is displayed.

(12) Log in to Google calendar in your browser: http://calendar.google.com.

(13) Experiment with the calendar and contacts integration by moving existing appointments, and make sure the contacts shown are updated.

(14) Please answer the questionnaire: http://www.surveymonkey.com/s/androidHomeScreen.

After the tasks were completed, we provided each user with a questionnaire consisting of 11 statements, consisting of three different parts, dealing with the *web application*, *context-awareness*, and *cloud computing,* respectively, in which participants indicated their opinions on a 4-point Likert scale anchored with *strongly disagree (SD)/disagree (D)/agree (A)/strongly agree (SA)*. Evaluation results are summarized in Table 2.

Responses regarding the web application showed that the web application performed as expected and information was successfully pushed to the device through 3G or WLAN. In terms of context-aware information, a clear majority supported a tight integration with the Google cloud services and a very positive bias was registered towards calendar and contacts integration. The users actively expressed wishes to see integration with more cloud-based services. Further into opinions on cloud computing, the participants indicate a mixed attitude towards cloud vulnerability and cloud data storage. The participants appreciate data being stored in the cloud and using this as part of the data foundation for the application, but the scepticism is shown through comments regarding security and trustworthiness of the storage of the data. Currently this is regarded as an interesting aspect but is out of scope for this project. For future work this will be important to investigate in terms of user satisfaction and possible adoption.

5. Analysis and Discussion

From the literature we point at the ability for modern applications to adapt to their environment as a central feature [5, 10, 11]. Edwards [24] argued that such tailoring of data and sharing of contextual information would improve user interaction and eliminate manual tasks. Results from the user evaluation support this. The users both find it attractive and have positive attitudes towards automation of tasks such as push updates of information by tailoring the interface. This work has further elaborated on context-aware integration and has shown how it is possible to arrange interplay between device context-aware information, such as sensors, and cloud-based context-aware information such as calendar data, contacts, and applications building upon suggestions for further research on adaptive cloud behaviour as identified in [16, 17, 25].

To register the tags the standard Google Calendar and Contacts web-interface were used. Such a tight integration with the Google services and exposure of private information was not regarded as a negative issue. As shown in the results, most of the users surveyed disagreed that this was an inconvenience. This perception makes room for further integration with Google services in future research, where, amongst them, the Google+ platform will be particularly interesting as this may bring opportunities for integrating the social aspect and possibly merge context-awareness with social networks.

Sensors are an important source of information input in any real world context and several previous research contributions have looked into this topic. The work presented in this paper follows in the footsteps of research such as that of Parviainen et al. [26] and extends sensor integration to a new level. By taking advantage of the rich hardware available on modern smartphones, the developed application is able to have tighter and more comprehensively integrated sensors in the solution. Although sensor integration as a source for context-awareness is well received, it still needs to be further enhanced. In particular it would be useful to find out appropriate extent and thresholds that should be used for sensor activation and deactivation. We have shown that it is feasible to implement sensors and extend their context-aware influence by having them cooperate with cloud-based services in a cross-source web application scenario. Further research includes investigating sensor thresholds and the management of different sources by different people in a web scenario.

6. Conclusion

In this paper we investigated context-aware and cloud-based adaptation of mobile devices and user's experience. Our research has added a new and novel contribution to the area of context-awareness in the cloud setup. We have proposed and demonstrated principles in implemented applications, whereby context-aware information is harvested from several dimensions to build a rich foundation on which to base our algorithms for context-aware computation. Furthermore, we have exploited and combined this with the area of cloud computing technology to create a new user experience and a new way to invoke control over user's mobile phone, one of the main features of which is liberating the user from manual configurations of mobile phones simply by having preferred options stored in a cloud-based configuration file which is pushed to any access device at logon. Through a developed application suite, we have shown the feasibility of such an approach, reinforced by a generally positive user evaluation. Moreover, we believe our solution, incorporating remote and automatic configuration of Android phone, advances the research area of context-aware information.

In further research, it would be interesting to investigate security, especially in terms of protecting user data and user perceptions thereof. We will continue to add more cloud services to our framework and expand on the exploitation of multidimensional context-awareness to facilitate seamless configuration and adaptation and use implicit user's needs.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] M. Mowbray and S. Pearson, "A client-based privacy manager for cloud computing," in *Proceedings of the 4th International ICST Conference on Communication System Software and Middleware (COMSWARE '09)*, pp. 5:1–5:8, ACM, Dublin, Ireland, June 2009.

[2] Y. Shen, M. Wang, X. Tang, Y. Luo, and M. Guo, "Context-aware HCI service selection," *Mobile Information Systems*, vol. 8, no. 3, pp. 231–254, 2012.

[3] M. Strobbe, O. Van Laere, F. Ongenae et al., "Novel applications integrate location and context information," *IEEE Pervasive Computing*, vol. 11, no. 2, pp. 64–73, 2012.

[4] L. M. Vaquero, L. Rodero-Merino, L. Caceres, and M. Linder, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[5] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and Ubiquitous Computing: First International Symposium, HUC'99 Karlsruhe, Germany, September 27–29, 1999 Proceedings*, vol. 1707 of *Lecture Notes in Computer Science*, pp. 304–307, Springer, Berlin, Germany, 1999.

[6] T.-M. Grønli, J. Hansen, G. Ghinea, and M. Younas, "Context-aware and cloud based adaptation of the user experience," in *Proceedings of the 27th IEEE International Conference on Advanced Information Networking and Applications (AINA '13)*, pp. 885–891, Barcelona, Spain, March 2013.

[7] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA '10)*, pp. 27–33, IEEE, Perth, Australia, April 2010.

[8] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, Version 15, National Institute of Standards and Technology, Information Technology Laboratory, 2009.

[9] Y. Vigfusson and G. Chockler, "Clouds at the crossroads: research perspectives," *Crossroads*, vol. 16, no. 3, pp. 10–13, 2010.

[10] M. Alhamad, T. Dillon, C. Wu, and E. Chang, "Response time for cloud computing providers," in *Proceedings of the 12th International Conference on Information Integration and Web-based Applications and Services (iiWAS '10)*, pp. 603–606, November 2010.

[11] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the weather tomorrow?: towards a benchmark for the cloud," in *Proceedings of the 2nd International Workshop on Testing Database Systems (DBTest '09)*, pp. 9:1–9:6, ACM, Providence, RI, USA, June-July 2009.

[12] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, "Interconnecting federated clouds by using publish-subscribe service," *Cluster Computing*, vol. 16, no. 4, pp. 887–903, 2013.

[13] C. Paniagua, S. N. Srirama, and H. Flores, "Bakabs: managing load of cloud-based web applications from mobiles," in *Proceedings of the 13th International Conference on Information Integration and Web-Based Applications and Services (iiWAS '11)*, pp. 485–490, Ho Chi Minh City, Vietnam, December 2011.

[14] A. Cidon, T. M. London, S. Katti, C. Kozyrakis, and M. Rosenblum, "MARS: adaptive remote execution for multi-threaded mobile devices," in *Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld '11)*, Cascais, Portugal, October 2011.

[15] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville, "The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise," *Software: Practice and Experience*, vol. 42, no. 4, pp. 447–465, 2012.

[16] L. Mei, W. K. Chan, and T. H. Tse, "A tale of clouds: paradigm comparisons and some thoughts on research issues," in *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC '08)*, pp. 464–469, Yilan, Taiwan, December 2008.

[17] L. Mei, Z. Zhang, and W. K. Chan, "More tales of clouds: software engineering research issues from the cloud application perspective," in *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '09)*, pp. 525–530, IEEE, Seattle, Wash, USA, July 2009.

[18] Gartner, "Gartner Identifies the Top 10 Strategic Technologies for 2012," 2011, http://www.gartner.com/it/page.jsp?id=1826214.

[19] R. Kamal, *Mobile Computing*, Oxford University Press, New York, NY, USA, 2008.

[20] M. Pohja, "Server push with instant messaging," in *Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC '09)*, pp. 653–658, March 2009.

[21] H. Flores, S. N. Srirama, and C. Paniagua, "A generic middleware framework for handling process intensive hybrid cloud services from mobiles," in *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia (MoMM '11)*, pp. 87–94, Ho Chi Minh City, Vietnam, December 2011.

[22] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[23] S. L. Kiani, A. Anjum, N. Bessis, R. Hill, and M. Knappmeyer, "Energy conservation in mobile devices and applications: a case for context parsing, processing and distribution in clouds," *Mobile Information Systems*, vol. 9, no. 1, pp. 1–17, 2013.

[24] W. K. Edwards, "Putting computing in context: an infrastructure to support extensible context-enhanced collaborative applications," *ACM Transactions on Computer-Human Interaction*, vol. 12, no. 4, pp. 446–474, 2005.

[25] J. H. Christensen, "Using RESTful web-services and cloud computing to create next generation mobile applications," in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA '09)*, pp. 627–634, Orlando, Fla, USA, October 2009.

[26] M. Parviainen, T. Pirinen, and P. Pertilä, "A speaker localization system for lecture room environment," in *Proceedings of the 3rd international conference on Machine Learning for Multimodal Interaction (MLMI '06)*, pp. 225–235, Bethesda, Md, USA, May 2006.