International Journal of Advanced Computer Research (ISSN (print):2249-7277 ISSN (online):2277-7970)
Volume-3 Number-2 Issue-10 June-2013

Improved registration system of Cloudtarun over Android Devices

Tarun Goyal[1], Aakanksha Agrawal[2], Somil Jain[3], Vaibhav Doshi[4]

Abstract

In this paper we have shown the some improvement in the registration system of the Cloud based application cloudtarun for Android Devices. We examine a technology called Cloud to Device Messaging and how well it integrates with cloud computing. In our investigation we look at the performance of the library, integration with Google App Engine and also the development tools including the API. We had done the performance tests of our application Cloudtarun and to make user registration system easy & simpler, we introduce a new open source library we call Simple-Cloudtarun.

Keywords

Android Emulators, Google Cloud Platform, Cloudtarun & Eclipse Indigo.

1. Introduction

Mobile phones, and especially smartphones, are becoming increasingly popular. Gartner reported that in 2010 the smartphone sales to end-users increased by 72.1%, while mobile device sales as a whole increased by 31.8% to a total of 1.6 billion units sold [1]. Consequently, developers are able to reach a very large audience with new and innovative software. Another important development in mobile phone technology is the high-speed network available to the public. With EDGE, 3G and even 4G in some places, users are almost always connected. The feature-rich and powerful mobile devices combined with a high-speed network provide many new and exciting innovation possibilities.

This work was supported in part by the Department of Computer Science & Engineering, Shekhawati Group of Institutions, India.

TarunGoyal, Department of Computer Science/IT, Skekhawati Institute of Engineering & Technology, Sikar, Rajasthan, India.

Aakanksha Agrawal, Department of Information Technology, GBPUAT, Pantnagar, Uttarakhand, India.

Somil Jain, Department of Computer Science, Skekhawati Institute of Engineering & Technology, Sikar, Rajasthan, India.

VaibhavDoshi, Department of Computer Science/IT, Skekhawati Institute of Engineering & Technology, Sikar, Rajasthan, India.

One of these is cloud computing, which has emerged as a promising technology direction in the last few years, especially when utilized in combination with new smartphones. The term cloud computing refers to the applications delivered over the Internet specifically and the hardware and systems software that is providing these services [2].

Cloud computing will in many cases offer the appropriate scalability, flexibility and cost-model for many different types of services. According to the Gartner [3] both cloud computing and mobile applications and media tablets are on the top 10 list of strategic technologies for 2011. This proves the importance of these technologies and how crucial it is to continue to push the boundary on what is possible to offer to the end users with the ever-improving hardware and network infrastructure available. We believe that a combination of a cloud based content publisher and a smartphone content subscriber is an especially promising area in part due to all the benefits mentioned above. We will go into more detail about this later on in this paper.

The work presented in this paper focuses on a concept tying both messaging and user registration over application cloudtarun. Cloudtarun provides the possibility for mobile devices to receive messages from a content publisher via a cloud platform. Specifically, we are targeting integration of Google App Engine integrated with Android mobile devices. Accordingly, in this paper we will explore the improvement in the registration of Cloudtarun for the Cloud platform & Android Devices.

2. Related work

There are many examples of research exploring the cloud computing area. One example is a system called COSCA [5]. COSCA is a PaaS (Platform as a Service) system, designed for adaptability and modularity. The cloud service we chose to use is Google App Engine. App Engine makes it possible for developers to run their own applications on Google's infrastructure [6]. It is similar to the COSCA model in that it provides a PaaS system. This means that the platform, which for Google App Engine supports Java, Python and Go, has beenpre-configured and is maintained by the cloud provider.

We will go into more detail about the Google App Engine technology later in this paper.

Several large IT companies like Microsoft, Google and IBM all have initiatives relating to cloud computing [7]. This shows the importance and maturity of cloud-based technology, which is also highlighted by Gartner as one of the strategic technologies for 2011 [3]. The success of cloud computing is based on an economy of scale. The ultimate goal is to provide more, in particular the appearance of unlimited scalability and storage, for less money [8].

One particularly popular manifestation of cloud computing is in the form of Service Oriented Architecture (SOA). Since the term SOA was coined in 1996, it has become the state-of-the-art of software architecture thinking, and all large software vendors today offer various frameworks and implementations of SOA [9]. SOA is a framework for designing flexible and loosely integrated services, in e.g. distributed cloud environments. A key challenge when implementing SOA is that to achieve full effect, processes needs to be transformed into more loosely coupled services [10] and made available through cloud based systems. Thus, Tsai et al. [11] focuses on the same aspect by elaborating on the architectural side of SOA. They look into how this can enable processes and services to connect and reconnect in an agile manner during an operational modus such as runtime.

In our case study we will present a performance test of Cloud based Application Cloudtarun, testing out the integration of App Engine with Android in practice. There are other research efforts that have focused on performance evaluations of cloud computing specifically. Alhamad et al. [12] has done a performance evaluation of the Amazon EC2 service. Amazon EC2 is quite different from the PaaS model utilized by Google App Engine. EC2 gives the users the ability to control nearly the entire stack, from the kernel and upwards [2]. In the benchmarking test of EC2 they measured the response times every two hours during several days. The main contribution from this paper is the testing of the isolation across the same hardware of virtual machines, which are hosted by a cloud provider. This experiment is quite different from ours, both in the cloud infrastructure and the fact that we are investigating the integration of a mobile client.
Mei et al. [7] highlight 4 main research areas in cloud computing which are particularly interesting; 1) Pluggable computing entities, 2) Data accesstransparency, 3) Adaptive behavior of cloud applications, and 4) Automatic discovery of application quality. In our work the investigation of Adaptive behavior of cloud applications is one of the main research areas. We will provide an experience report of both the push messaging technology on the Android platform and how it integrates with cloud computing.

We will provide an initial performance test in this paper to have an insight into the Cloudtarun response times. There are several previous research efforts focusing on cloud computing performance. A paper by Binning et al. [8] present some ideas on what a benchmarking test should look like for cloud computing. They argue that the benchmark should highlight issues like the adaptability of a system (the ability to adapt to changing load in terms of scalability and cost), run the tests from different locations and access "Web 2.0"-like applications including multimedia content. These pointers are provided for more general-purpose benchmark tests than what we will include in this paper, where we specifically focus on the integration of cloud platform and Cloudtarun in Android. Still, there are some important ideas from this paper. Specifically, we see the importance of including performance of the system under different loads and running the tests from different locations. More detail is presented in section 5 where we include information on how an extended performance test should be conducted in future work.

We can summarize the contributions of this paper in two main categories; 1) we will provide an experience report of the state of the art of application Cloudtarun on the Android platform and the integration of cloud computing. 2) After presenting the standard tools and libraries, we will introduce our attempt at improving development of registration system of application on Android.

Our research tries to answer two main questions:

- How well does the Android Devices perform over Google Cloud Platform?
- Are there parts of the library that does not work well? How can we improve them?

We will give a more in-depth look at the technologies explored in the next section. Then we will present the details of our implemented system and explain how our experiment was conducted.

3. Cloudtarun on android devices

We have decided to focus on Android in our work because it is an open platform that is well suited for experimentation. Also the GWT is especially interesting because it integrates well with the Google infrastructure, mainly their cloud platform (App Engine).

A. The Android Operating System

The Android platform is released under the Open Handset Alliance [14], the goal of which is to create open standards for mobile devices. Android is an open source mobile operating system based on the Linux kernel. For application development, Android facilitates the use of 2D and 3D graphic libraries, a customized SQL engine for persistent storage, advanced network capabilities such as Edge, 3G and WLAN, a whole range of sensor services and other features such as near field communication (NFC), Bluetooth and voice control. The API is constantly evolving and the next release (4.0, Ice cream sandwich) is a huge step forward in terms of available features from earlier releases, now also bridging the gap between the phones and the tablets.
Before the standard cloud based feature was available for Android it was common to use a polling mechanism. This meant that the applications would constantly poll the server for updates, much like POP mail clients work where the clients will send a request to the server asking if it has any updates [15]. Polling for updates has several well-known drawbacks, including the major challenge of setting the frequency of the requests sent to the server. This issue is especially important when working with mobile devices since developers have additional issues like battery life and network coverage and cost to deal with.

B. Cloudtarun Integration with the Cloud

In addition we integrate cloudtarun with the cloud using Google App Engine. Google App Engine is a PaaS technology that enables developers to easily create highly scalable and flexible applications. Developers using Google App Engine are presented with a pre-configured platform and they must create applications within the limitations imposed by this platform. Each application runs in a secure environment that provides limited access to the operating system [6].

The limitations mostly focus on keeping the application stateless. Developing server-side applications using state is by most developers considered to be bad practice and we believe that the majority applications should be able to live in the Google infrastructure aslong as they take certain precautions. If developers do not need the missing features or are able to work around them, which should be possible in most cases, this platform presents some very interesting benefits, including high scalability.

One major benefit of utilizing the Google infrastructure is the ability to take advantage of Google provided APIs for authentication and sending emails using Google accounts. There is no need to implement a separate user management system with application-specific user details. The system can easily use standard Google accounts. It is worth noting that many applications will not be able to run in the cloud because it handles sensitive data of some sort. Indeed, there are many non-technical reasons why a cloud-based option may not be a choice at all for certain applications.

4. Simple-cloudtarun

We started experimenting with Cloudtarun quickly after its initial release in 2012. This means it is in an experimentation phase and there is a possibility it will be changed before the final version is released. Since the technology is still in an early stage of development there is room for improvement. We are presenting an alternative and simplified version of the Cloudtarun API in this paper that we believe is a step in the right direction.

The Cloudtarun technology tries to connect the mobile platform with cloud computing, where the idea is that the cloud should be able to contact the mobile application without the need for a custom polling mechanism.

Android Cloud to Device Messaging was launched in Android 2.2 and was implemented to make it easier for mobile applications to sync data with servers [4]. Several of the standard Google applications, like Gmail, Contacts and Calendar, use GWT. From the 2.2 release of Android this was also made available to third-party developers [15].

The system is only meant to send short messages notifying the mobile application that new and updated data can be retrieved from the server; in so doing, it is not designed to handle large amount of data. To use Cloudtarun the application must register the mobile device by signing in with their Google

account. From the Google Cloud service a registration id will be generated. This id needs to be sent to the server application, which will include it when sending out push messages to target a specific device. Figure 1 shows a simplified presentation of how Cloudtarun works. Both the register device and server authentication have to be completed before any messages can be sent to the one device to another device using the Google Cloud Platform by using Cloudtarun application

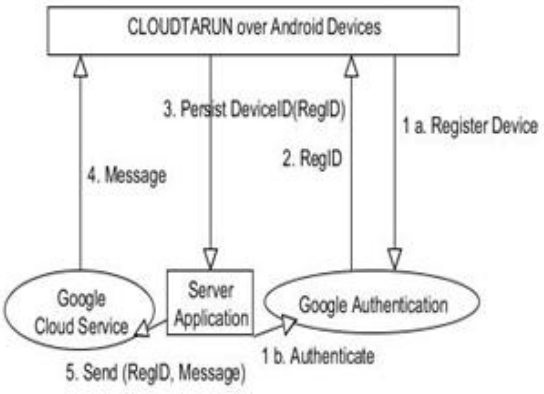

Figure 1: Cloudtarun

The Cloudtarun library has several similarities with the previously mentioned Minstrel experiment [13]. Both use the publish/subscribe model, where subscribers must register at the content publisher to receive content. Similar to Cloudtarun, Minstrel was also extended to include support for a Cloud based messaging system for mobile devices. So that database and message system can work over Cloud Platform completely and properly.

When developing applications using GWT there are a few required tasks for the developer. Permissions and other service-related entries must be added to the *AndroidManifest.xml* file. Also, the limited Java API offered by Google consists of a few base classes that you need to extend. There is no help from these classes to deal with registration ids, as these needs to be sent from the client to the server application.

We wanted to make the integration between the application and Cloudtarun simpler. It is our experience that the current API is both missing important features like registration id management and its abstraction is too low level. We also wanted to add more flexibility to the system, each developer should not have to deal with the same problems over and over again because of an APIthat is not very user-friendly. We therefore introduce an attempt at improving the development of Cloud messaging on Android with a library we call Simple-Cloudtarun [16]. To develop and improve the registration system of the Cloudtarun there are some of the Key features to be used which are explained below as follows.

A.Key Components

The system we have implemented is built using three major components (figure 2):

- Manifest generator (outside the core library)
- Server utilities

Android library

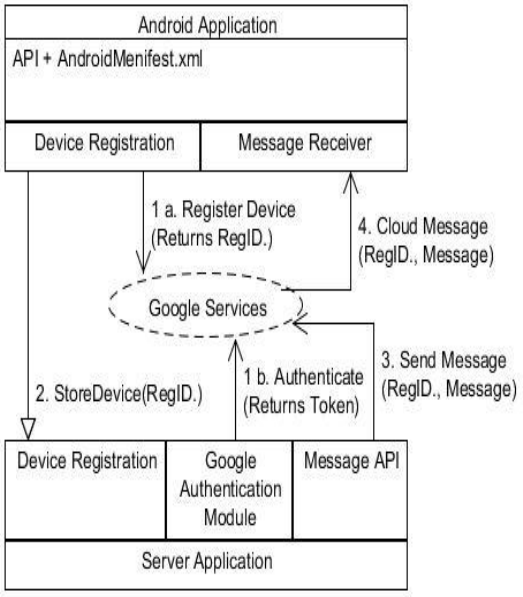

Figure 2: Simple-Cloudtarun

The manifest generator was created as an App Engine application, presenting a webpage for generating the *AndroidManifest.xml* file. Eliminating the need for the error prone manual task it was to write this by hand. All the developers need to do is to enter the package name of the application in the text field. When the generate button is pushed this will generate the necessary xml code that must be added in the project manifest file.

On the server side we wanted to add helper classes for dealing with the common scenarios in a cloud messaging system. This includes receiving and extracting registration ids, generating authentication tokens and sending messages. This library was specifically created to be independent of any server side technology. It should not require the server application to be running on App Engine for example; this is very different from the way Google

has solved this with their Eclipse plugin, which we will look closer at in section 6.

The main part is the library we created for the Android client. It handles all the common tasks of working with Cloudtarun: registering the device, sending registration id and receiving messages. It extends the *cloudtarun.jar* file created by Google, but tries to hide the complexity and low abstraction level of that library. It offers both annotation-based and direct callback options for the developer to use. Using annotations gives the most flexibility and is in our opinion the best way of providing the flexibility and simplicity in this system.

5. Application and development

To complete the performance test we needed to create a server and client application. We also used this system to try out the Simple-Cloudtarun API and see how it compared to working with the standard API.

A. Sample Application

The sample application consisted of a server application hosted on App Engine and an Android client. It sent requests over a period of 200 minutes, with one request every minute. The client application was responsible for keeping the state of the requests and measured the time used and the server application simply answered the sent requests.
The basic functionality of the sample application:

- On startup, initialize both the Android and the server applications. The Android application receives the registration id, while the server application obtains an authentication token to be able to send messages. The registration id is also sent to the server since all push messages are sent to a specific registration id.
- The Android client sends a ping request and waits for a HTTP 200 response. This is done to ensure that the server application was up and running and ready to respond.
- The server application receives the request and responds when it is ready.
- Now the performance test starts, the Android application records the current time and then sends a request to the server application that it should send a push message back to the phone.
- The server application receives the request and then immediately sends out a push message to the client.

When this push message is received at the Android application, it will again record the current time. Now it can calculate the total time used.

6. Results and discussion

In this case study we wanted to test two issues with Cloudtarun. Firstly we wanted to do some initial performance tests of the technology in order to examine its potential and how well it would perform in a simple benchmarking test. Secondly, we wanted to compare thestandard Cloudtarun library with our custom Simple-Cloudtarun API.

A. Performance

In the performance test, we completed a small test over 200 minutes, sending one request per minute. Google states that it does not provide any guarantees when it comes to delivering messages, and the performance data we recorded is only an initial test. More thorough tests need to be done to get a good comparison, including testing from different locations, devices, networks and also compare the Cloudtarun performance with other similar technologies. Figure 3 shows the detailed results

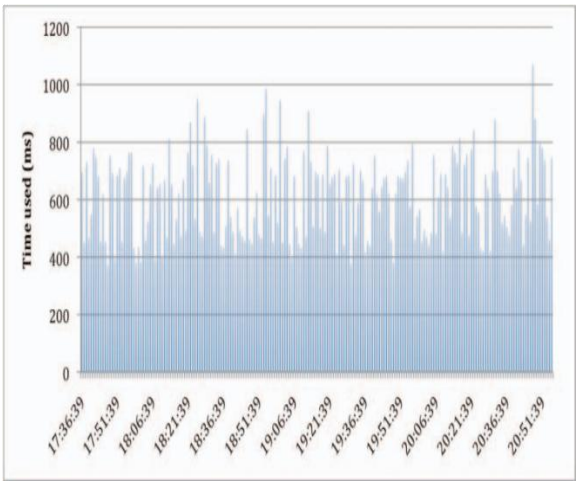

Figure 3: Cloudtarun performance test

Thus, the average time used in our test was 610 ms., this includes the extra request we needed to send to trigger the message. The response times were within a standard deviation of 145 ms. In comparison we did a simple ping command to the server application deployed on App Engine. Over a series of 200 ping commands we recorded an average value of 47ms. This means the actual time used for the push messages averages under 600ms if the initial request

time is deducted. In our test we used HTC Evo on a WIFI network.

There were several limitations of the test and a further investigation would need to test on several different devices and device types, networks, times and locations. By extending the experiment with these new features we will be able to provide more reliable performance measurements. Also, by including other push-based technologies on the Android platform we will be able to a more comprehensive comparison of these technologies.

B.Comparison, Cloudtarun vs Simple-Cloudtarun

When developing Cloudtarun applications there are a few base classes offered by Google that will need to beused in the application. For registering and receiving push messages or error messages the receiver class needs to extend a class called CloudtarunBaseReceiver (figure 4).

```
public class Receiver
extendsCloudtarunBaseReceiver {
publicSCloudtarunReceiver() {
super("email@test.com");
   }

@Override
protected voidonMessage(Context context,
        Intent intent) {}

@Override
public voidonError(Context context,
        String errorMsg) {}

@Override
public voidonRegistered(Context context,
        String registrationId) throws IOException {}
}
```

Figure 4: Standard Cloudtarun code

This has a few disadvantages. Firstly, in Java you cannot extend more than one class, so you are tied to the BaseReceiver-class. Secondly, you must strictly follow the methods you override. Both method names and input parameters must be exactly the same.

In Simple-Cloudtarun we wanted to improve this API by making it simpler to develop with. Firstly we added a direct callback mechanism. Instead of inheritance it uses an interface that the developer must implement in his own code. This is not very different from the base receiver class, but one avoids the problem of single inheritance since we provided an interface instead of an implementation. The second alternative we implemented was using annotations. This greatly adds to the flexibility of the code. By adding *@OnRegistered*, *@OnMessage* and *@OnError* in the code at method level, we could call these methods via reflection. This had the benefits of both being very easy to read and understand, but also giving the developers the flexibility of calling their methods whatever they wanted and even the choice of input parameters. It also makes it possible to split the implementation across several classes. We added logic to support input with the Context object or for example just the registrationId String. That means if your application did not need the Context object in your callback, you didn't have to add it to your codebase. At this point, the main drawback we know about in our solution is that reflection does impact performance. Inour initial tests it did add a maximum 1ms overall increase in the time used. This should not be a problem in any real world scenarios, but further investigation of the overall performance needs to be done.

On the server side we wanted to add a few simple utility classes. Specifically we wanted to support: authentication (generating Google security tokens), receiving registration ids (simple parsing of requests) and message generation/sending. One of the main goals of this library was to make it server independent. We created classes not dependent on any specific server vendor, and although not extensively tested yet, you should even be able to run it on your desktop computer.

The new feature not available in the Google library was handling of registration ids. We added support for automatic sending of the registration id in a background thread from the client to the server application. By adding a server URL in the *@OnRegistered* annotation (figure 5), it would automatically trigger this feature.

```
@OnRegistered("http://server-url")
public voidregistered(String registrationId) {}

@OnMessage
public voidmessage(String msg) {}

@OnError
public voiderror(String errorMsg) {}
```

Figure 5: Simple-Cloudtarun API

The main goals we had was a simple library that would make it easier to use Cloudtarun and not tie down the developer in any predefined technology choice. This is quite different from the Eclipse plugin Google is working on for Cloudtarun. The Google plugin for Eclipse [17] contains a wizard for auto generating a project containing the basis for a Cloudtarun project connected to Google App Engine. This is very different from our library by using code generation and being directly tied to technologies including App Engine, GWT (Google Web Toolkit) and several other Google specific libraries.

7. Conclusion and future work

For the first question we investigated what the development tools and libraries are for standard Cloudtarun. We also implemented a test application where we conducted a small performance test to try out the technology. In our opinion the Cloudtarun technology is particularly interesting because it integrates push technology on the Android platform with cloud computing.

We did find room for improvement in the standard API offered by Google. We have tried to improve this in our own open source project called Simple-Cloudtarun. The project is still under development and has not been released in a final version. More testing and experimentation is needed before it is released to the community in a stable version.

In future work we want to further expand on the performance test. We would like to include other technologies like XMPP (Extensible Messaging and Presence Protocol) and MQTT (Message Queue Telemetry Transport). The test would also benefit from having the commercial Cloudtarun libraries, mainly Urban Airship [18] and Xtify [19], included in the performance tests and general investigation. This would give a complete report on the status of push messaging on Android.

Last but not least, working towards a final release of Simple-Cloudtarun is also something we would like to continue with in future work.

References

[1] Gartner, "Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010; Smartphone Sales Grew 72 Percent in 2010," Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010; Smartphone Sales Grew 72 Percent in 2010.

[2] M. Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing," 2009.

[3] Gartner, "Gartner Identifies the Top 10 Strategic Technologies for 2011," Gartner Identifies the Top 10 Strategic Technologies for 2011. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1454221.

[4] Google, "Android Cloud to Device Messaging Framework," Android Cloud to Device Messaging Framework.

[5] S. Kächele, J. Domaschka, and F. J. Hauck, "COSCA: an easy-touse component-based PaaS cloud system for common applications," in Proceedings of the First International Workshop on Cloud Computing Platforms, New York, NY, USA, 2011, pp. 4:1–4:6.

[6] Google, "What Is Google App Engine?," What Is Google App Engine?[Online]. Available: http://code.google.com/appengine/docs/whatisgo ogleappengine.htm l.

[7] Lijun Mei, W. K. Chan, and T. H. Tse, "A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues," in IEEE Asia-Pacific Services Computing Conference, 2008. APSCC '08, 2008, pp. 464-469.

[8] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the weather tomorrow?: towards a benchmark for the cloud," in Proceedings of the Second International Workshop on Testing Database Systems, New York, NY, USA, 2009, pp. 9:1–9:6.

[9] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, Applied SOA: Service-Oriented Architecture and Design Strategies, 1st ed. Wiley, 2008.

[10] H. Demirkan, R. J. Kauffman, J. A. Vayghan, H.-G. Fill, D. Karagiannis, and P. P. Maglio, "Service-oriented technology and management: Perspectives on research and practice for the coming decade," Electron. Commer. Rec. Appl., vol. 7, no. 4, pp. 356–376, Dec. 2008.

[11] W. T. Tsai, R. A. Paul, C. Fan, and X. Wei, "Dynamic Architectures ForSoa-Based Applications," J. Integr. Des. Process Sci., vol. 11, no. 4, pp. 65–105, Dec. 2007.

[12] 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 Webbased Applications & Services, New York, NY, USA, 2010, pp. 603–606.
[13] T. Goyal, A. Singh and A. Agrawal," Cloudtarun: Application Simulated over GAE using Android Emulators" in IJCA, vol. 57, no. 4, pp. 26-31, Nov. 2012.
[14] Open Handset Alliance, "Open Handset Alliance," Open Handset Alliance. [Online]. Available: http://www.openhandsetalliance.com/. [Accessed: 13-Oct-2011].
[15] T. Bray, "Android Cloud to Device Messaging," Android Cloud to Device Messaging. [Online]. Available: http://androiddevelopers.blogspot.com/2010/05/android-cloud-to-devicemessaging.html. [Accessed: 13-Oct-2011].
[16] J. Hansen, "Simple-C2DM," Simple-C2DM.[Online]. Available: http://code.google.com/p/simple-c2dm/. [Accessed: 14-Oct-2011].
[17] Google, "Google Plugin for Eclipse," Google Plugin for Eclipse. [Online]. Available: http://code.google.com/eclipse/. [Accessed: 14-Oct-2011].
[18] Urban Airship, "Urban Airship," Urban Airship. [Online]. Available: http://urbanairship.com/. [Accessed: 14-Oct-2011].
[19] Xtify, "Xtify," Xtify. [Online]. Available: http://xtify.com/. [Accessed: 14-Oct-2011].

I am **Tarun Goyal**,M.Tech CSE(H) from BTKIT, Dwarahat, Uttarakhand & B,Tech IT(H) from GECB, Bikaner, Rajasthan. Presently I am working as Assistant Professor CSE/IT, SGI, Sikar, Rajasthan, India. My area of researches are cloud computing, Web technology, data structures. I have published around 10 research papers in international journals (including IEEE, Elsevier, etc).

I am **Aakanksha Agrawal** ,M.Tech CSE(H) from BTKIT, Dwarahat, Uttarakhand & B,Tech CSE(H) from AITS, Halwani, Uttarakhand. Presently I am working as Teaching Personnel IT, GBPUAT, Pantnagar, Uttarakhand, India. My area of researches are Networking, Cloud Computing & Mobile Computing Technology. I have published around 10 research papers in international journals till date (including IEEE, Elsevier, etc).

I am **Somil Jain**, M.Tech CSE (H) from Jagannath University, Jaipur, Rajasthan & B,Tech CSE from SGI, Sikar, Rajasthan. Presently I am working as Assistant Professor CSE/IT, SGI, Sikar, Rajasthan, India. My areas of researches are Networking, C language & Data Structures. I have published around 02 research papers in international journals till date.