Towards Mobile Cloud Applications: Offloading Resource-intensive Tasks to Hybrid Clouds

Huber Flores, Satish Narayana Srirama, Carlos Paniagua
Institute of Computer Science, University of Tartu
J. Liivi 2, Tartu, Estonia
{huber, srirama, paniagua}@ut.ee

Abstract

Cloud computing becomes mobile when a mobile device tries to access the shared pool of computing resources provided by the cloud, on demand. Mobile applications may enrich their functionality by delegating heavy tasks to the clouds as the remote processing and storage have become possible by adding asynchronous behavior in the communication. However, developing mobile cloud applications involves working with services and APIs from different cloud vendors, which mostly are not inter-operable across clouds. Moreover, by adding asynchronicity, mobile applications must rely on push mechanisms which are considered to be moderately reliable, and thus not recommended in scenarios that require high scalability and QoS. To counter these problems, a middleware framework, Mobile Cloud Middleware (MCM) is designed, which handles the interoperability issues and eases the use of process-intensive services from smartphones by extending the concept of Mobile Host. MCM is developed as an intermediary between the mobile and the cloud, which hides the complexity of dealing with multiple cloud services from mobiles. Several applications are presented to show the benefits of mobiles going cloud-aware. Moreover, to verify the scalability of MCM, load tests are performed on the hybrid cloud resources using well known load balancing mechanisms like HAProxy and Tsung. From the study we found out that it is possible to handle hybrid cloud services from mobiles by using MCM. The analysis demonstrated that the MCM shows reasonable performance levels of interaction with the user, thus validating the proof of concept. Finally, MCM fosters the utilization of different types of cloud services rather than the traditional mobile cloud services based on data synchronization. By offloading heavy tasks to the clouds, the framework extends the processing power and storage space

capabilities of the constrained smart phones. The applications mentioned in the paper bring an added value by being success stories for mobile cloud computing domain in general.

Index Terms

Mobile Cloud Computing; Middleware; Mobile Cloud Services; Hybrid Clouds; SOA

I. Introduction

Mobile computing and cloud computing [1] domains are converging as the prominent technologies that enable developing the next generation of ubiquitous services based on data-intensive processing. Cloud computing is a style of computing in which, typically, resources scalable on demand are provided "as a service (aaS)" over the Internet to users who need not have knowledge of, expertise in, or control over the cloud infrastructure that supports them. The provisioning of cloud services can occur at the Infrastructural level (IaaS) or Platform level (PaaS) or Software level (SaaS). Cloud computing mainly forwards the utility computing model, where consumers pay on the basis of their usage. Moreover, cloud computing promises the availability of virtually infinite resources.

On the other hand, improvements in smartphones, on hardware (embedded sensors, memory, power consumption, touchscreen, better ergonomic design, etc.), in software (numerous and more sophisticated applications possible due to the release of iPhone [2] and Android [3] platforms), in transmission (higher data transmission rates achieved with 3G and 4G technologies) and in Wifi networks ubiquity, have contributed towards having higher mobile penetration and better services provided to the customers. Now the mobile users can collaborate and share information with considerable ease.

Mobiles enter cloud computing domain by trying to access the shared pool of computing resources provided by the cloud on demand. Mobile technologies are drawing the attention to the clouds due to the demand of the applications, for processing power, storage space and energy saving. This has lead to the Mobile Cloud Computing (MCC) domain. Applications that benefit from such a Mobile Cloud are from different domains like social networks, location based services, context-aware systems etc. [4]

To counter the problems with the interoperability across multiple clouds, to perform data-intensive processing invocation from the handset and to introduce the platform independence fea-

ture for the mobile cloud applications, we have designed a Mobile Cloud Middleware (MCM) [5]. The middleware provides a unique interface (REST based) for mobile connection and multiple internal interfaces and adapters, which manage the connection and communication between different clouds so that the offloaded tasks from the mobile can be processed by different clouds.

While multiple approaches have been focused on offloading code components from the mobile to nearby server as processing is needed [6], [7], in this paper, we focus on enriching the mobile applications by offloading data that requires high amounts of computational processing (e.g. sensor and image processing) and by delegating multiple mobile tasks to different cloud architectures (e.g. activities in sequence or parallel). In this context, MCM and the resource intensive tasks can easily be envisioned in several scenarios. For instance, Zompopo [8], consists of the provisioning of context-aware services for processing data collected by the accelerometer with the purpose of creating an intelligent calendar. CroudSTag [9], consists of the formation of a social group by recognizing people in a set of pictures stored in the cloud. Finally, it is shown how MCM can help in managing the cloud resources themselves with the Bakabs [10] application. This paper presents an overview of Zompopo and Bakabs applications.

Since most of the reasonable mobile cloud services require significant time to process the request; it is logical to have asynchronous invocation of the mobile cloud services. Asynchronicity is added to the MCM by using push notification services provided by different mobile application development platforms and by extending the concept of Mobile Host [11]. Once the MCM prototype was developed, it was extensively analyzed for its performance and the details are addressed in this paper. MCM also improves the quality of service (QoS) for mobiles and helps in maintaining soft-real time responses.

The rest of this paper is organized as follows. Section 2 introduces the Mobile Cloud Middleware concept with realization details. Section 3 discusses the asynchronous invocation of cloud services. Section 4 addresses the applications that benefit from MCM. Section 5 explains the considered performance model with detailed performance analysis. Section 6 discusses the scalability of the MCM. Section 7 addresses the related work and section 8 concludes the paper with future research directions.

II. Related Works

Middleware approaches similar to MCM have been addressed in the literature. MCCM (Mobile Cloud Computing Middleware) [12] is a project which involves the use of a middleware, standing between the mobile and the cloud, for the consumption of web services (WS) [13] in a mobile mashup WS application. It handles creating user profiles from the context of the mobile phone, storing the system configuration (pre-defined set of WS which can be consumed from the handset) and the service configuration respectively, for managing existing resources in the cloud. However, such middleware and the API support seem to be tightly coupled in contrast to our proposed solution. Moreover, from studying the applications developed with MCCM we observed that it is not suitable for service invocations that require resource-intensive processing like in Zompopo or Bakabs, due to the lack of support for asynchronicity.

Cloud agency [14] is another solution that aims to integrate GRID, cloud computing and mobile agents. The specific role of GRID is to offer a common and secure infrastructure for managing the virtual cluster of the cloud through the use of mobile agents. Agents introduce features that provide the users a simple way for configuring virtual clusters. However, such solution does not explain how agents enable keeping soft-real time responses when a cloud service is invoked from the mobile or how the communication with different cloud providers is handled by the GRID. Agora [15] is another middleware solution which is under development, which will enable new large-scale mobile-cluster applications and will use mobile devices as nodes of a large-scale cloud-computing infrastructure. Agora will enable the devices to work together seamlessly. However, a concrete implementation cannot be found yet.

However, MCM is more scalable and portable solution, when compared to other solutions, as it manages all the transactions with multiple clouds in a transparent way. It allows building lighter applications for mobile devices, as the developers do not have to deal with several APIs. The applications just have to implement the functions provided as a web service at the middleware. MCM mainly enables interoperability across multiple cloud architectures.

While extensive research has been conducted on push technologies [16]–[18] and its need in mobile communication, the asynchronous notification feature of MCM using Mobile Host really separates it from other approaches. Mobile Host asynchronous communication frees the mobile resources during most of the invocation process and it is not limited to mobile platform, size or

number of the data packages which can be pushed to the mobile.

III. Mobile Cloud Middleware

Mobile cloud services use the shared pool of computing resources provided by the clouds to get the process and storage intensive tasks done from smart phones. Generally mobile applications focus at enriching their functionality to a mashup application, using the cloud services. However, clouds are looking forward to the mobile domain, having their expectations focused in the idea of data synchronization services. Mobile sync refers to the synchronization of data in the handset with a server and a portal in the cloud. This kind of approach takes advantage of the cloud allowing the centralization of data resources (contacts, e-mail, pictures, etc.) in the storage service, from several sources (social networks, e-mail providers, etc.), for being accessible via SyncML [19] protocol, in such a way that empowers the managing of real-time information from the handset. Some of the most popular vendors offering synchronization services are Funambol [20], everdroid.com, rseven.com and Memotoo.com.

Even though pure mobile cloud services are based on data synchronization, there are also other cloud services in each layer of the cloud computing domain (IaaS, PaaS, and SaaS) that might enable the mobiles for increasing their functional capabilities. These services mainly offer processing data-intensive tasks that are high demanding for a handset. This can be observed as most of the SaaS are extended to fit the mobile demand. For example, both Google docs and Zoho provide mobile versions of their office suites, also Picasa and flickr offer services to visualize pictures that fit the mobile screen.

While several mobile cloud services are existing today, most of them are bounded by numerous constraints like the mobile platform restrictions, cloud provider's technology choices etc. They provide proprietary APIs and routines to consume the services. Therefore, cloud interoperability is not possible and when a lighter mobile application is to be created, it has to be developed for a specific cloud provider. For example: an application created on Android using jets3t [21] to access S3 from Amazon [22], and Walrus from Eucalyptus [23] has to suffer some changes in its configuration when it tries to access each of the cloud storage services. Even though Eucalyptus is compatible with Amazon infrastructure, there is no full integration between them. Moreover, cloud vendors are generally observed to be slow in providing APIs for multiple mobile platforms. For example, at the time of writing this paper, Amazon has just released the mobile API for both

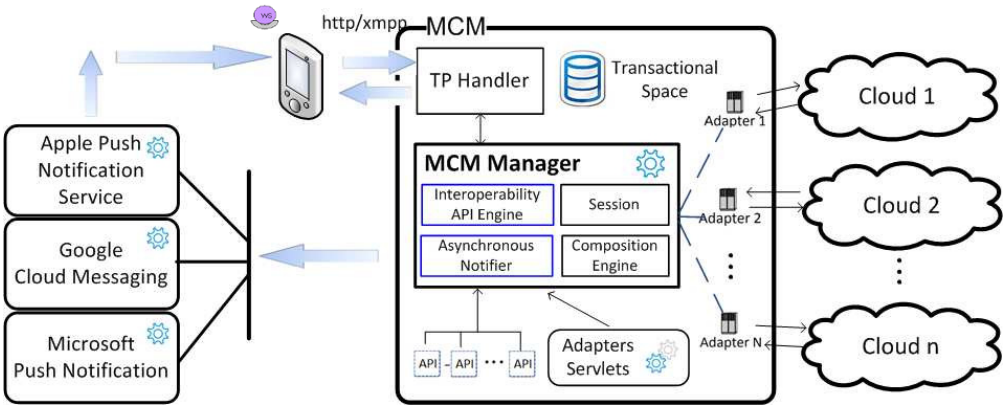

Fig. 1: Architecture of the Mobile Cloud Middleware

Android and iOS platforms, however, they are still in beta. To address most of these problems, we have developed the Mobile Cloud Middleware (MCM).

A. MCM Architecture and Realization

MCM is introduced as an intermediary between the mobile phones and the cloud. The architecture is shown in figure 1. MCM fosters mobile platform heterogeneity and the combination of different cloud services into a mobile mashup application. When an application tries to connect to a basic cloud service, it connects to the TP-Handler component of the middleware, which receives the request. The transportation handler can receive the requests based on several protocols like the Hypertext Transfer Protocol (HTTP) or the Extensible Messaging and Presence Protocol (XMPP) [24]. The request is then processed by the MCM-Manager for creating the adapters that will be used in the transactional process with the clouds. Figure 2a shows MCM-Manager components logic and behavior, figure 2b shows a general behavior of an adapter.

When the request is forwarded to the MCM-Manager, it first creates a session (in a transactional space) assigning a unique identifier for saving the system configuration of the handset (OS, clouds' credentials, etc.) and the service configuration requested (list of services, cloud providers, types of transactions, etc.) in a session space, respectively. The identifier is used for handling different requests from multiple mobile devices and for sending the notification back when the process running in the cloud is finished. The transactional space is also used for exchanging data between the clouds (to avoid offloading the same information from the mobile, again and

again) and manipulating data acquired per each cloud transaction in the composition process. Based on the request, the service transaction is managed by the Interoperability-API-Engine or the Composition-Engine, in order to decide the set of Web APIs for handling the hybrid cloud services.

Once the interoperability-API-engine decides which API set it is going to use, the MCM Manager requests for the specific routines from the Adapter-Servlets. The servlets contain the set of functions for the consumption of the cloud services. Finally, MCM-Manager encapsulates the API and the routine in an adapter for performing the transactions and accessing the SaaS. The result of each cloud transaction is sent back to the handset in a JSON (JavaScript Object Notation) [25] format, based on the application design.

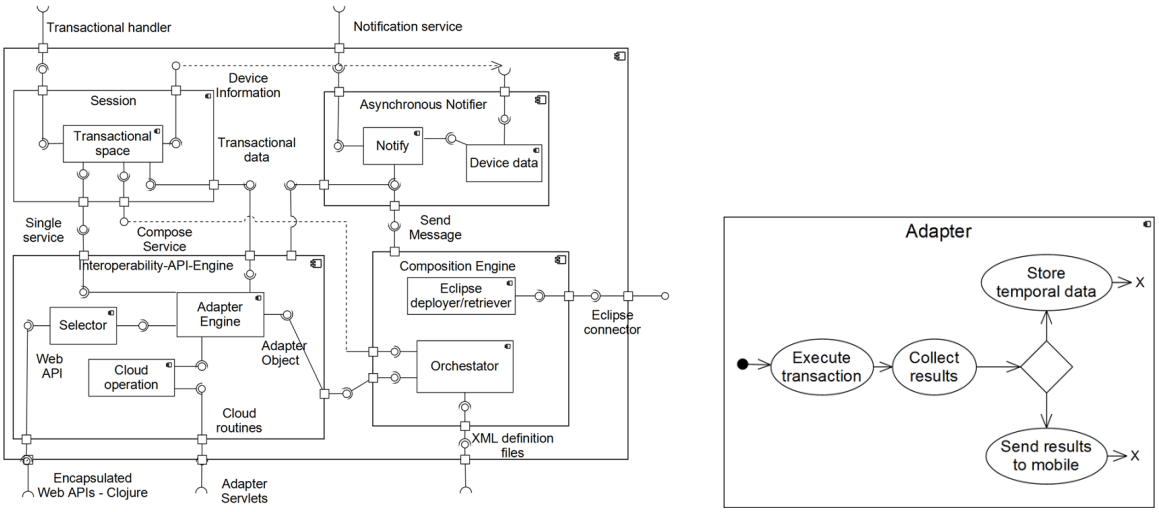

Fig. 2: (a) MCM-Manager Component Diagram in UML 2.0 (b) General Behavior of an Adapter

Since the completion of a cloud task, most often, is time consuming, it is not logical to make the mobile application waiting for getting a response back. Moreover, such waiting time is not tolerable for the user and mobile application usability perspective. To counter this, MCM supports asynchronous mobile cloud service invocation for both Android and iOS platforms, and it is discussed in detail in section 3.

MCM is implemented in Java as a portable module based on Servlets 3.0 technology, which can easily be deployed on a Tomcat Server or any other application server such as Jetty or GlassFish.

Mobile cloud services using APIs from Amazon EC2, S3, Google and some open source cloud projects like Eucalyptus are considered. From the cloud infrastructure level, jets3t is the API within MCM that enables the access to the storage services from Amazon and Google. Jets3t is an open source API that handles the maintenance for buckets and objects (creation, deletion and modification). A modified version of the API was implemented for handling the storage service of Walrus (Eucalyptus storage service). Latest version of jets3t handles synchronization of objects and folders from the cloud. Typica API and the Amazon API are implemented to manage (turn on/off, attach volumes etc.) the instances from Eucalyptus and EC2 respectively. MCM also has support for SaaS from facebook [26], Google and face.com [27].

IV. Asynchronicity for Handling Process Intensive Mobile Cloud Services

Some mobile applications, whose functionality depends on mobile cloud services, can provide a reasonable response time to the user. For example, searching for a location in Google Maps or requesting for a preview of a picture in flickr. However, when a mobile application needs to perform a task which is expected to be time consuming (e.g. tasks based on MapReduce [28] data analysis), it cannot hang the mobile phone until the response arrives. Mobile devices tend to get stuck if the computation offloading requires long waiting, and in some cases the OS just kills the task when it senses that its low on memory (Android case), or just do not allow performing another task at the same time.

As a solution to address these issues, MCM implements an asynchronous notification feature for mobile devices that foster the de-coupling between the client and server. In the asynchronous process, when a mobile application sends a request to the middleware, the handset immediately gets a response that the transaction has been delegated to remote execution in the cloud, while the status of the mobile application is sent to local background. Now the mobile device can continue with other activities. Once the process is finished at the cloud, a notification about the result of the task is sent back to the mobile, so as to reactivate the application running in the background, and thus the user can continue the activity.

Asynchronicity is added to the MCM by extending the concept of Mobile Host and by implementing push notification services, specific to Android and iOS platforms, due to their popularity in the mobile market, using GCM (Google Cloud Messaging for Android) [29] and APNS (Apple Push Notification Service) [30], respectively. Moreover, MCM also supports sending notification

to Android based on AC2DM (Android Cloud to Device Messaging Framework) [31].

A. MCM and GCM

GCM is the enhanced notification service provided by Google for sending asynchronous messages to Android devices. It has been released as replacement for the deprecated AC2DM service and includes new features such as unlimited message quota, decoupling of the mobile device from a Google account for receiving message data (devices running Android 4.0.4 or higher) and message throttling that enables to prevent sending a flood of messages to a single handset, among others.

Basically, a mobile application that implements the GCM mechanism for receiving messages, first, has to register itself against the GCM server for acquiring a registration ID. Messages are sent to the mobile using this identifier from the application server, which lasts till the mobile explicitly unregisters itself, or until Google refreshes the GCM servers. An API key is required for the application server, in order to pass message data to the GCM service. This key is generated by the developer using the Google APIs console and it is used as the sender ID of the message.

MCM sends a message to the mobile by sending the registration ID, the API key and the payload to the GCM servers, where the message is enqueued for delivery (with maximum of 5 attempts) or stored in the case the mobile is offline. GCM allows up to 100 messages stored before discarding it. Once, the device is active for receiving the message, the system executes an Intent Broadcast for passing the raw data to the specified Android application. GCM do not guarantee the delivery of a message and the sending order. Messages with payload can contain up to 4K of data and are encapsulated in a JSON format.

Since GCM is an enhanced version of AC2DM, an application can easily be migrated from one mechanism to other. The overall communication process is depicted in figure 3.

B. MCM and APNS

Similar to above mechanism, Apple devices running iOS 3.0 or newer can also receive asynchronous messages provided through APNS. APNS messages are sent through binary interface (gateway.push.apple.com:2195, gateway.sandbox.push.apple.com:2195) that uses streaming TCP socket design. Forwarding messages to device happens through constantly open IP connection. TLS (or SSL) certificate (provisioned through iOS developer portal) is needed for creating secure

communication channel and for establishing trusted provider identity. To avoid being considered a DoS (Denial of Service) attacker, using one connection for multiple notifications, rather than creating new connection for each notification, is desired.

APNS has a feedback service that records failed notification delivery attempts. This information should be checked regularly to avoid sending messages to devices that do not have the targeted application installed anymore. For the same reason application should register itself at MCM for notifications at each start by providing at least it's device token that it has received from APNS. Device token is a 32 byte hexadecimal number that is unique for an application on a device and does not change.

Messages sent to iOS devices via APNS consist of JSON payload with maximum length of 256 bytes. Within message payload, values for keys alert, sound and badge can be used to customize the message alert being shown to user upon receiving it. Because the payload size is limited, it is used to provide enough information for the application to make request for additional details. When the device is unable to receive notifications for some time (e.g. due to being offline or switched off) and multiple notifications have been sent, only the last one is guaranteed to be delivered. Figure 3 also illustrates the asynchronous mechanism using the APNS.

C. MCM and Mobile Host

While most of the relevant mobile platforms are providing asynchronous mechanisms (aka notification services) for dealing with remote executions, the mechanisms have certain constraints and limitations such as being platform specific (e.g. GCM/AC2DM for Android, APNS for iOS, INS for WebSphere EveryPlace, etc.), the size of the message that can be pushed into the device (e.g. 1024 bytes for Android, 256 bytes for iOS, etc.) and the number of the messages that can be sent to a single handset (e.g. 200,000 for AC2DM). Moreover, such mechanisms are considered to be moderately reliable, and thus are not recommended in scenarios that require high scalability and quality of service. For example: AC2DM simply stops retrying after some delivery attempts.

As a solution to address these issues, MCM implements an asynchronous notification feature, which is a tricky process. To have a generalized solution, one can make the mobile check for the status of the service regularly. However, this puts a lot of load on the mobile networks. Alternatively, we can make the mobile device a service provider, which was studied in mobile

web service provisioning project [32]. The study developed a Mobile Host [33] which can be used in providing web services from the smart phones. When the smart phone acts as a server, the mobile cloud service response can be sent directly to the device. The current implementations of the Mobile Host are available only for PersonalJava and J2ME platforms, which we have extended for Android platform.

Mobile Host for Android is based on REST protocol, where the services are considered as resources that can be accessed through HTTP requests. Android SDK provides a mechanism to establish Server Sockets communication between the device and the clients; consequently, the HTTP request can be handled in the device. The services exposed by Mobile Host can be categorized in two types namely OSGI Services and Messaging Services. The OSGI services run over Apache Felix (an OSGI implementation for Android), where all the services run as bundles that can be easily deployed dynamically. Such services also implement a Java interface enforcing them to handle all the HTTP methods. The Messaging Services, interact with the applications installed in the device, passing data between the Mobile Host and the applications. MCM utilizes Messaging Services for reactivating the application as it is waiting for the cloud results.

Mobile Host exposes its services through ZeroConf, which consists of a set of techniques for automatic configuration and creation of a usable local Internet protocol network. ZeroConf dynamically configures the host in the network assigning them an IP address and also a domain name. Furthermore, ZeroConf provides a mechanism for service discovery and domain resolution. Mobile Host for Android uses JmDNS, a service discovery protocol which is an implementation of ZeroConf. JmDNS assigns a local domain name to Mobile Host which can be used by other devices to access the services exposed by the host. Moreover, JmDNS is totally compatible with other implementations of ZeroConf for other platforms such as Bonjour for Apple.

Figure 3 shows the invocation process in detail. When a mobile application sends a request to the middleware, the handset immediately gets a response that the transaction has been delegated to remote execution in the cloud, while the status of the mobile application is sent to local background. Now the mobile device can continue with other activities. Once the process is finished at the cloud, MCM sends a HTTP POST request for the resource http://mobilehost/notification (Messaging Service), with two parameters, application and message, where application corresponds to the mobile application that needs to be notified about the results and message is a JSON string with the results of the invocation. Mobile Host resolves the request, reactivates the

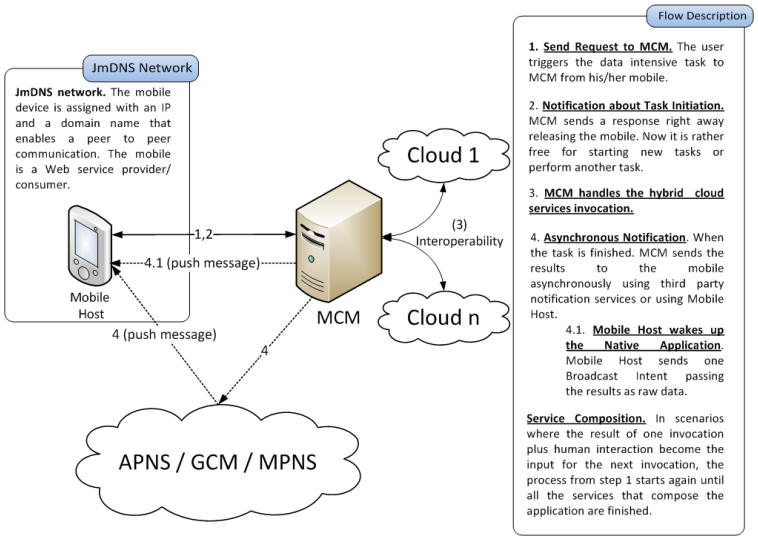

Fig. 3: MCM asynchronous communication using push notification technologies and Mobile Host

application running in the background by sending a message through a Broadcast Intent, and thus the user can continue the activity. The approach can also be used to concurrently execute several tasks in multiple clouds.

V. Towards Mobile Cloud Applications

MCM aggregates the resources from multiple clouds and thus encourages a development approach to enrich the functionality of mobile applications. Two scenarios are described here, to show the potential in combining hybrid cloud services from multiple sources within a handset.

A. Zompopo: Mobile Calendar Prediction using the Accelerometer and Cloud Services

Zompopo is a context-aware application that tries to extend the capabilities of a generic calendar adding a feature that makes use of the accelerometer sensor for predicting the activities which the user will perform during the day based on the recognition of previous week's activities (figure 4 shows the scenario in detail). The accelerometer is able to track information for the recognition of multiple human activities (walking, running etc.); each activity is differentiated according to the entropy data collected by the accelerometer as shown in figure 5.

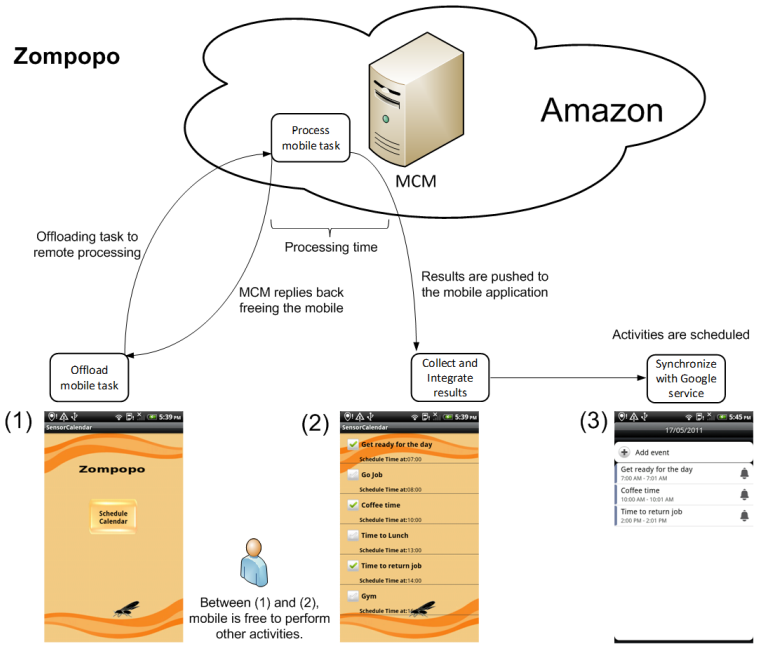

Fig. 4: Zompopo application flow

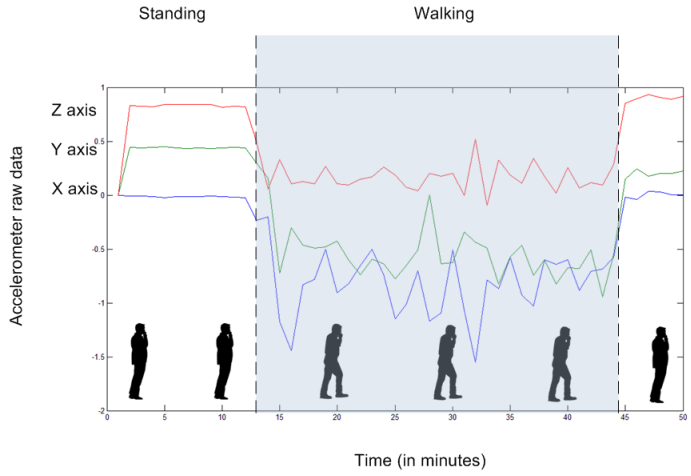

Fig. 5: 3-axes readings for different activities

When the mobile application gets started, it remains executing in the handset background; the data from the accelerometer is gathered and stored in a SQLite [34] database. The accelerometer provides information across time related with acceleration along x, y and z axes. Therefore, acceleration can be sensed in three directions, forward/backward, left/right and up/down. For example: in the case of a runner, up/down is measured as the crouching when he/she is warming

up before starting to run, forward/backward is related with speeding up and slowing down, and left/right involves making turns while he/she is running.

By default the accelerometer is always sensing environmental changes that are appended to the database file and then offloaded from the mobile to the cloud storage once per day (end of the day). When Zompopo attempts to schedule the user's activities (beginning of the day), it sends a request to MCM for processing the set of historical sensor data that has been collected. The remote processing at the cloud, consists of applying a MapReduce algorithm (Hadoop), which is shown in figure 6 to each file for finding the times in which the user was actively moving, and then matching the times of the complete set for finding the most frequent hours of labors. Based on this information, Zompopo infers the activities that the user will perform again.

The MapReduce algorithm consists of two steps, the Map and the Reduce functions. The Map function takes one set of key value pairs and maps them to intermediate key pairs. The intermediate key pairs are sorted and grouped together by the framework and passed to the reduce function. The Reduce function takes the intermediate key pairs and produces the output. Each line within the file contains the following information < indexi, ti, xi, yi, zi>, where ti represents a timestamp taken during different hours of the day, xi, yi and zi are the 3 axes measured by the accelerometer at time ti. Those data is mapped individually to one key value with the following structure < ti, [xi, yi, zi] > to produce one Sequential File that is the input for the MapReduce process. Since the prediction is based on the x axis, the Map function takes each key < ti, [xi, yi, zi] > from the Sequential File and creates one temporary key < ti, xi >.

Later, the Reduce function receives the temporary keys grouped and sorted by time. Each key represents one set of x values and each key is processed by one Reducer. Two statistical measures, the mean and the standard deviation are used for analyzing the data, and thus determining whether the user is moving or not. The standard deviation indicates how the data is spread out over a range of values. Based on figure 5 the more spread out the values are the more the user moves and in the opposite way the more the values are close to each other the less the user is moving. The Reduce function calculates the two statistical measures and uses the standard deviation to determine if the user is moving or not, for the given set of values. One threshold value for the standard deviation is defined with a value of 1 for this experiment. If the standard deviation is below the threshold values the algorithm infers that the user was not moving. If the standard deviation is greater than the threshold value it means that the data is spread enough to infer that

Algorithm 1 Classification algorithm using MapReduce

Require: : csvFile, inputFile, outputFile, threshold

```
    Map from csvFile to sequenceFile
        writeNumber(csvFile, sequenceFile)
            writter.append(time, num)
    Generate intermediate keys from sequenceFile
        map(LongWritable time, DoubleWritable x, Context context)
            context.write(time,x)
    Reduce - calculate statistical measures and infer the movement
        reduce(LongWritable key, Iterable<DoubleWritable> values, Context context)
            context.write(time,x)
    Calculate Mean values
        mean = calculateMean(values)
    Calculate Standard Deviation
        sd = calculateStandardDeviation(values, mean)
    Determine the value of the movement
if (sd > threshold) then
        output[MOVEMENT] = MOVE_ACTIVITY
else
        output[MOVEMENT] = NOT_MOVE_ACTIVITY
end if
    Generate final keys
        context.write(key, new ArrayWritable(output))
```

the user was moving by the time the data was measured. The Reduce produces the output in CSV file with information such <time during the day, Accelerometer Measure, Standard Deviation, Action>, where accelerometer measure is the mean value of the x values received by the Reducer and Action is the activity inferred by the algorithm.

Since the processing is time consuming, MCM makes use of the asynchronous notification feature for releasing the mobile OS from the server and for sending a notification back altogether

with the result when the task has finished at the cloud. Once the handset is notified about the results, Zompopo shows a screen with the list of suggested activities (hour + name of the activity) that could be included in the daily calendar. Since Zompopo was developed for Android; the activities are created using the default calendar application which comes with the OS. The android calendar allows to use SyncML for the synchronization with Google calendar service, and thus changes are replicated to the cloud calendar service automatically. However, the creation of activities is also possible from MCM as is able to access Google cloud services. Therefore Zompopo is not just tied to the mobile platform and can easily be extended.

B. Bakabs: Managing Load of Cloud-based Web Applications from Mobiles

Not just the proper utilization of cloud services, with MCM, we can also think of applications which can help in managing the cloud resources themselves. Bakabs is an application, developed for Android and iOS devices, which makes use of cloud services such as Google Analytics to track the traffic of web-sites, replicated on multiple instances in different locations. Bakabs also suggests the number and type of instances from each region that are required to handle the loads, based on the linear programming model. Based on the suggestions, the user can decide to turn on/off instances, thus saving costs taking advantage of the pay-as-you-go model and the elasticity of the cloud.

Similar to Zompopo, the application uses several hybrid cloud services. Google Analytics provides the number of visits and the average page load time per profile, where each profile corresponds to a web application. The details are then sent to the respective public and private clouds to get the exact deployment configuration. The next service takes the loads as input and predicts the optimum cluster configuration using its own linear programming model. For accessing the Analytics Reports the GData API is used, and for accessing the cloud-hosting services the Amazon and typica APIs are used. Finally, the response is sent back to the handset in a JSON format to be presented to the user.

Once the results are shown by the mobile application, the user can decide whether to launch or to stop cloud instances. Since launching/stopping instances in the cloud is usually a time consuming task, this action is triggered from the device but is delegated to MCM which monitors the progress and informs the user, through an asynchronous notification message, when the task is finished. Figure 6 shows the screenshots of Bakabs.

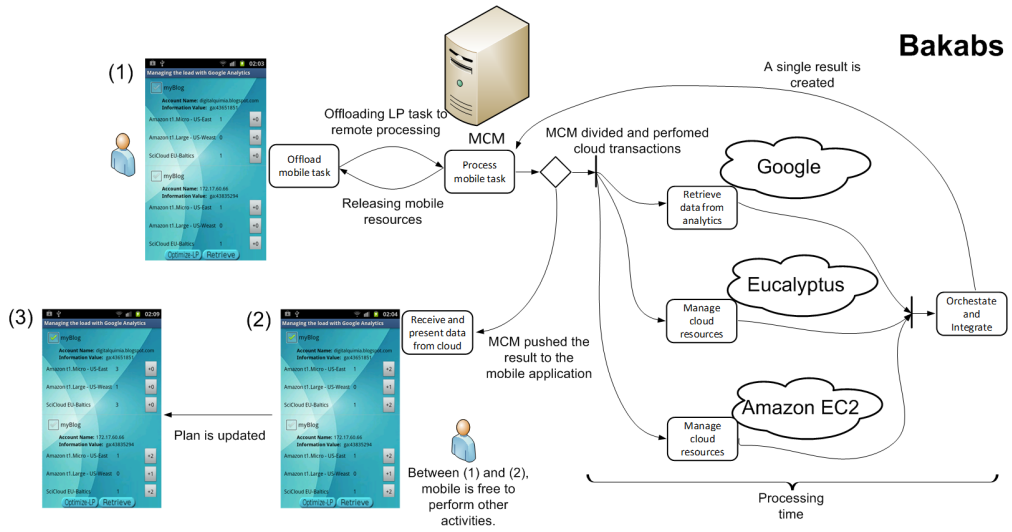

Fig. 6: Bakabs application flow

1) Bakabs' Model: Linear Programming on the Cloud: In order to fit better their costumers' needs, Cloud Hosting providers offer different types of instances with different configurations and prices. For example, Amazon offers a vast spectrum of instance types combining the size and the purpose of the instances. Based on the size, the Amazon's customers can choose from Micro to Extra Large instances which are approximately eight times more powerful than Small instances. Each instance type has its own software and hardware configuration and the price varies from one type to another. Similarly, the price of cloud resources varies from one region to another region. For instance, computing resources in Amazon AWS in US-East are cheaper than in Asia Pacific-Tokyo. However, there are several factors such as latency and cost of data transmission that might affect the cost-effectiveness of running instances in a specific region. Therefore, instead of using just one type of instances it is more cost efficient to use a combination of instances types and regions. Bakabs' aim is to find the optimal number of cloud instances of each type in each region at the minimum cost that still satisfies the workload.

The number of servers needed to satisfy the current load in the web applications is estimated using linear programming techniques. Linear Programming is a mathematical method whose aim is to determine the optimal solution for a given mathematical model. This mathematical model consists of a linear objective function that needs to be optimized and it is subject to a set of constraints expressed as equalities or inequalities. The constraints define a finite space

of solutions, known as feasible region, where the objective function can be evaluated. A linear programming algorithm finds a point inside the feasible region where the objective function is optimal, whether maximum or minimum, and still the constraints are satisfied. The objective function and constraints are composed by variables and parameters. The parameters are known values that are set before the execution of the linear programming algorithm. The algorithm's aim is to maximize or minimize the objective function by assigning values to the variables of the problem.

The parameters of the Bakab's linear programming model include:

- $c_{t,r}$, cost of the instance type t running in the region r. In the case of type of instances, there are three types, namely Amazon Large (amazonL), Amazon Micro (amazonM) and SciCloud [35] Large (sciCloudL). Similarly, six regions can be considered, namely US-East (USE), US-West (USW), EU-Ireland (EUI),Asia Pacific-Singapore (APS), Asia-Pacific-Tokyo (APT) and EU-Baltics (EUB). The model would have one paramater $c_{t,r}$ for each of three types and for each region available (i.e. $t \in T$ ={amazonL,amazonM,sciCloudL} and $r \in R$ ={USE,USW,EUI,APS,APT,EUB}).
- $p_{t,r}$, number of request that an instance of type t running in the region r can resolve during a minute.
- B, the maximum budget constraint.
- CC, number of instances that the cloud is capable to launch. Generally, private clouds are able to manage a limited number of instances at a time. Similarly, public clouds such Amazon AWS can launch at most 20 instances on demand, if up to 20 instances need to be launched the customer has to fill a form requesting for more resources.
- W, the value of the current workload in the system in requests per minute. This parameter is derived from the Google Analytics reports.

The variables correspond to the number of instances of different types running in different regions to be run:

- $x_{t,r}$, the number of cloud instances of type r running in the region r

The model tries to minimize the cost of having x number of instances of type t running in the region r at a cost $c_{t,r}$ for each instance type and region. The object function is defined as a sum across all the instances types $t \in T$ and all the regions $r \in R$.

$$Min \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ti,rj} * c_{t1,rj} \quad (1)$$

Finally, the linear programming model comprises three sets of constraints:

- The workload constraint

$$\sum_{i=1}^{n} \sum_{j=1}^{m} x_{ti,rj} * p_{ti,rj} > W \quad (2)$$

- The budget constraint B

$$\sum_{i=1}^{n} \sum_{j=1}^{m} x_{ti,rj} * c_{ti,rj} < B \quad (3)$$

- Private cloud's capacity constraint:

$$x_{t,r} < CC_{t,r} \quad (4)$$

The workload constraint states the sum of all the capacity p across all the instances types t and all the regions r must be enough to satisfy the current workload W in the system. The budget constraint states the sum of all the cost c across all the instances types t and all regions r must be under the user's budget B. The private cloud's capacity constraint states the number of running instances of the type t in the region r must be less than the capacity CC of the cloud. In other words, the number of instances that need to be run must be under budget and not exceed the cloud's capacity, if any, and still satisfy the workload in the system.

For the public clouds the price of each instance type t in a region r is provided by the vendor and can be easily accessed. However, for private clouds the cost estimation depends of several factors and it is quite specific for each cloud. In order to estimate the cost of run one instance in SciCloud several assumptions, such as the priority of the research projects among others, have been done since the cost estimation for private clouds is out of the scope of this paper. It is assumed that the cost of run one instance in SciCloud is the equivalent to $0.05 per hour. The capacity p of the instances has been estimated using Tsung [36], a load testing tool. In an scenario where each request consist of a HTTP POST request, uploading one file of 512 Kb, the instances considered in this paper, Amazon Micro, Amazon Large and SciCloud Large, show performances of ≈470, ≈525 and ≈500 requests resolved per minute respectively.

The solution of the optimization problem described above is the final step performed by MCM before sending the results back to the device. MCM uses OptimJ [37], a java-based

modeling language for optimization, which is a propetary product capable to resolve several types of optimization scenarios. For example, for a traffic of 3000 request per minute over a web application which is currently running in 5 instances (1 amazonM in USE, 1 sciCloudL in EUB, 1 amazonL in EUI, 1 amazonL in AST and 1 amazonM in AST), the model proposes to launch three new instances of type amazonM in EUE at a minimum cost of $1.367 per hour.

VI. Performance Analysis: Cloud Services Invocation from Mobiles

From the applications shown in the earlier section, we can derive that it is possible to handle process intensive hybrid cloud services from the smart phones, via the MCM and using its asynchronous push notification support. We later evaluated the scenarios to have a detailed performance analysis. The performance model and the analysis are addressed here. Figure 8 shows the sequence of activities that are performed during the execution of the application. Here the total application duration i.e. the total mobile cloud service invocation time, is:

$$T_{mcs} \cong T_{tr} + T_m + \Delta T_m + \ Max_{i=1}^{n}(T_{te_i} + T_{c_i}) + T_{pn} \tag{5}$$

Notice that (5) considers T_pn, when MCM uses the push notification approach of third party services (Figure 8b). In contrast, MCM also can rely on Mobile Host approach (Figure 8a). In this case, T_mh is considered instead of T_pn as follows:

$$T_{mcs} \cong T_{tr} + T_m + \Delta T_m + \ Max_{i=1}^{n}(T_{te_i} + T_{c_i}) + T_{mh} \tag{6}$$

Where, T_{tr} is the transmission time taken across the radio link for the invocation between the mobile phone and the MCM. The value includes the time taken to transmit the request to the cloud and the time taken to send the response back to the mobile. Apart from these values, several parameters also affect the transmission delays like the TCP packet loss, TCP acknowledgements, TCP congestion control etc. So a true estimate of the transmission delays is not always possible. Alternatively, one can take the values several times and can consider the mean values for the analysis. T_m is the time taken to process the request at the middleware. ΔT_m is the minute extra latency added to the performance of the MCM, as the mobile is immediately notified with the acknowledgment. T_{te} is the transmission time across the Internet/Ethernet for the invocation between the middleware and the cloud. T_c is the time taken to process the actual

service at the cloud. $\cong$ is considered in the equation as there are also other timestamps involved, like the client processing at the mobile phone. However, these values will be quite small and cannot be calculated exactly. The $Max_{i=1}^{n}$ is considered for the composite service case, which involves several mobile cloud service invocations. The access to multiple cloud services actually happens in parallel in Bakabs application. In such a scenario, the total time taken for handling the cloud services at MCM, T_{Cloud}, will be the maximum of the time taken by any of the cloud services ($Max_{i=1}^{n}(T_{te_i} + T_{c_i})$). However, in a case where the access to cloud services happens in sequence, for example in CroudSTag case, the T_{Cloud} becomes $\sum_{i=1}^{n}(T_{te_i} + T_{c_i})$.

Similarly, T_mh and T_pn represent the push notification time, which is the time taken to send the response of the mobile cloud service to the device via Mobile Host and third party push technologies, respectively. While Tmcs may seem a bit higher, the phone is rather free to continue with its tasks, so not much load on it. This is possible only due to the support for push notifications at the MCM. The mobile phone just sends the request and gets the acknowledgment back. Actual response from the cloud is sent to the mobile asynchronously. Thus the delay perceived at the mobile rather stays constant however big the Tmcs may be.

A. Analysis and results

To analyze the performance, the mobile cloud applications explained in section 4 are considered. For Zompopo, Eucalyptus Walrus storage services are used for saving the information collected by the accelerometer. A historical set consisting in one week of accelerometer data (one file per day) was stored in a Walrus bucket. In the case of Bakabs, Eucalyptus EC2 services are used for hosting the web applications and to collect the traffic data. The Web applications installed in the cloud instances are configured to be tracked by Google Analytics cloud services. The Eucalytpus cloud instances were launched and stopped several times depending on the load reported by Google Analytics. HTC desire phone [38], with a 5 megapixel color camera with auto focus and flash was considered for the analysis. It has a CPU speed of 1GHz, 576 MB of RAM and storage that can be extended up to 32GB. The applications were developed based on the Android platform, compatible with Android 2.2 API or higher. Wifi connection was used to connect the mobile to the middleware or the cloud. So, test cases were taken in a network with an upload rate of 1409 kbps and download rate of 3692 kbps, respectively. However, as mentioned already, estimating the true values of transmission capabilities achieved at a particular

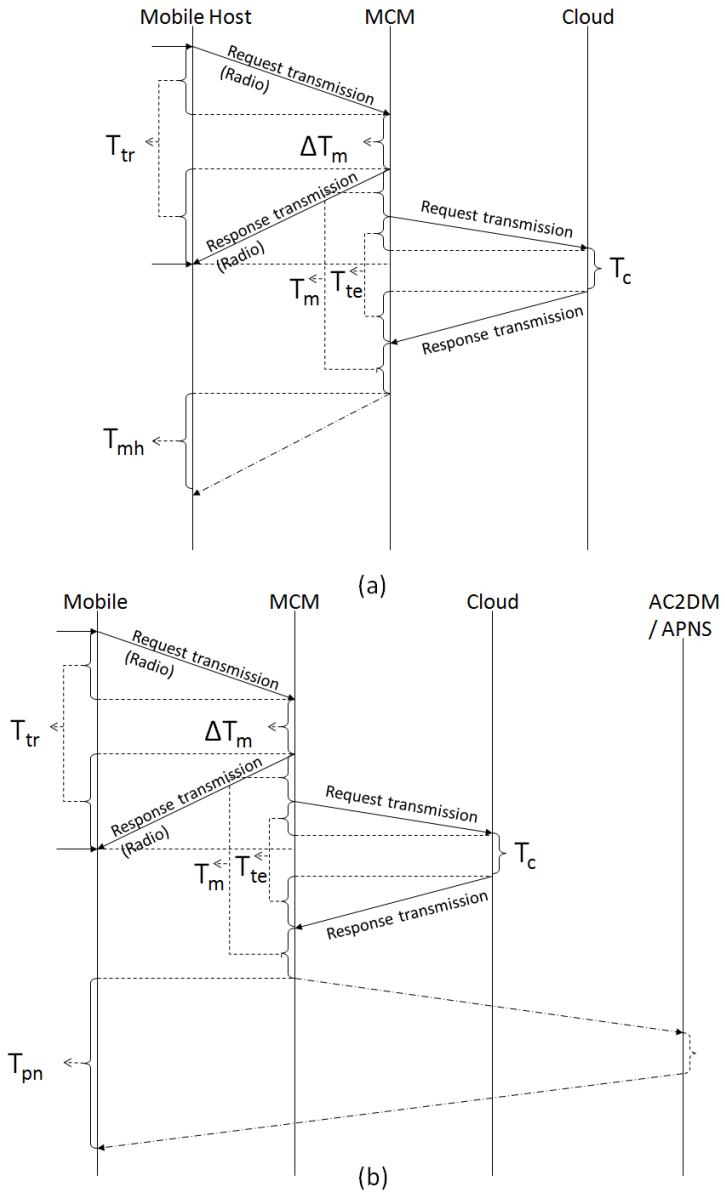

Fig. 7: Mobile cloud service invocation cycle: activities and timestamps in (a) Mobile Host case (b) Push notification services case

instance of time is not trivial. To counter the problem, we have taken the time stamps several times (5 times), across different parts of the day and the mean values are considered for the analysis.

The timestamps are shown in figure 8. The value of T_{tr} + ΔT_m is quite short (< 1 sec for both cases), which is acceptable for a mobile application and tolerable from the perspective of the

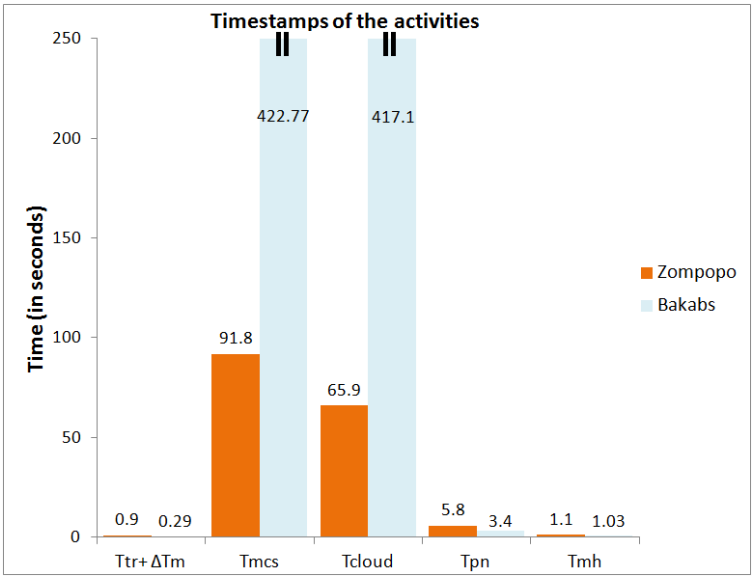

Fig. 8: Timestamps for Zompopo and Bakabs scenarios

user. Moreover, the user has the capability to start more data intensive tasks right after the last one or go with other general tasks, while the cloud services are being processed by the MCM. The total time taken for handling the cloud services at MCM,T_{Cloud} ($Max_{i=1}^{n}(T_{te_i} + T_{c_i})$), is also logical and higher as expected (70 sec for Zompopo, 410 sec for Bakabs). The T_{pn}varies depending on current traffic of the C2DM service and has an average of ∼5.1 seconds. Mobile Host based asynchronous approach is also considered for all the scenarios, as an alternative mechanism. The T_mh asynchronous time has an average of ∼1.2 seconds.

VII. Scalability of MCM

While MCM was successful in achieving the invocation of mobile cloud services from a handset and the interoperability between clouds, as a standalone framework, it faces the troubles with heavy loads. Scalability is quite relevant for mobile scenarios as the acceptable number of users that a mobile operator network is handling might increase unexpectedly (e.g. flash mobs or the popularity of a single mobile service etc.), and the solution should scale on demand. In order to fit such oscillating requirements, the middleware can be moved to the cloud, so that the MCM by default acquires the elasticity and horizontal scaling (achieving better QoS by adding more nodes to the cluster, rather than increasing capabilities of a single node) features of the

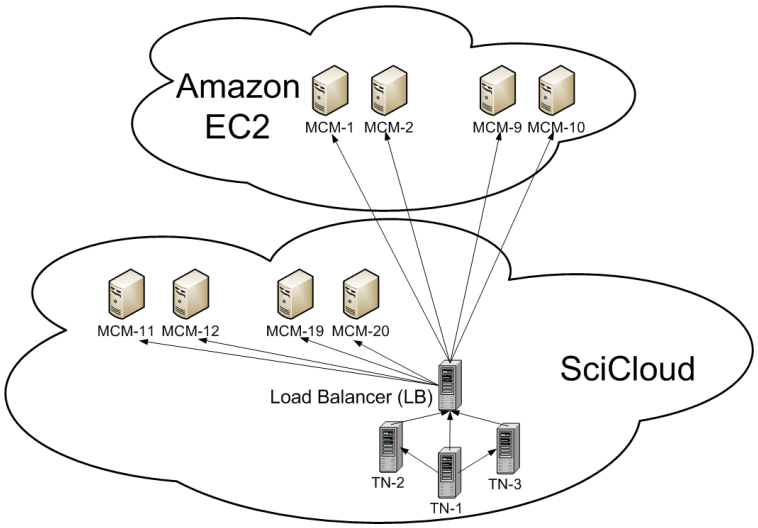

Fig. 9: Load test setup for the MCM

cloud.

To verify the scalability of the middleware, MCM was deployed on two different clouds (Amazon EC2 and SciCloud). Once the Amazon/SciCloud Machine Images (AMI, EMI respectively) are configured, MCM nodes can be added as needed for distributing the load among them with the load balancer (LB). Figure 9 shows the deployment scenario. The load balancing technique is based on the use of a proxy server in front of the MCM nodes. HAProxy [39] is considered as the LB as it allows dynamic behavior to the architecture and new MCM nodes can be added while the system is running (hot reconfiguration).

A. Scalability Analysis of MCM

Load testing of MCM was performed using Tsung (open source load testing software). Tsung was deployed in a distributed cluster composed of three nodes running on SciCloud (one primary and two secondary nodes). The primary node is in charge of executing the test plan and collecting all the results of each secondary node, so that information can be combined and analyzed into a single report using Tsung-plotter utility. The test plan is structured by blocks and consist of three parts, the server/client configuration part, in which the machines' information is defined. The load part that contains the information, it is related with the mean inter-arrival time between new clients and the phase duration. Here, the number of concurrent users is defined. For instance,

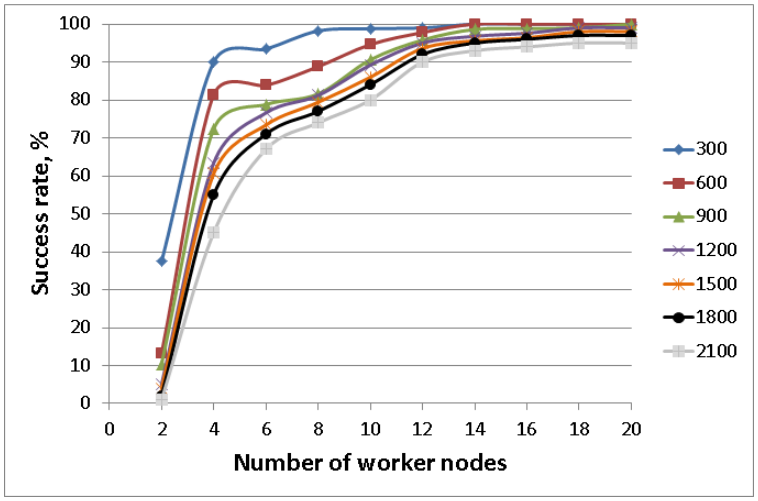

Fig. 10: Success rate of concurrent requests over multiple server nodes

for generating a load of three hundred users in one second a mean of 0.0033 was used (Tsung primary node divided the load into equal parts among the available Tsung nodes). And finally, The sessions part, in which the testing scenario that is conformed by the client requests (captured using Tsung-recorder) is configured. A single request consists of simulating the offloading of accelerometer sensor data to the cloud (Zompopo case). Uploading the file is requested by n concurrent threads, where n varied between 100 and 700 per Tsung node (TN). This makes 300 to 2100 on the load balancer. Thus, simulating a large number of simultaneous users connecting to the MCM.

On the cloud front, a load balancer and up to 20 MCM worker nodes were setup. To show the scale on demand of the solution, the number of server nodes were increased from 2 to 20. Since MCM nodes were divided 10 per each cloud, in each test, 2 new nodes were added (1 in Amazon EC2 and 1 in SciCloud). Servers running on Amazon EC2 infrastructure were using EC2 large instances. A large instance has 7.5 GB of memory and a CPU power of 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each). One EC2 compute unit is equivalent to a CPU capacity of 1.2 GHz XeonTM processor (CPU capacity of an EC2 compute unit do change in time). Servers running on SciCloud were using large instances. A large instance has 1.8 GB of memory and a CPU power equivalent to a CPU capacity of a 2.83 Ghz Intel®Core™Quad 2. Both load balancer and MCM nodes were running on 64 bit Linux platforms (SUSE in Amazon

and Ubuntu in SciCloud). The load balancer was setup for using Round-robin scheduling, so the load can be divided into equal portions among the worker nodes.

In the load test of the MCM, it was measured how the success rate of the requests depends on the number of framework nodes depending on the number of concurrent requests. A request is considered as success, if it gets a response back (i.e. transaction completed) before the connection or response timeout occurs (Since some of the worker nodes are located in Amazon, the timeout might be increased due network congestion or packet retransmission, and thus some connections may get lost). Similarly, the success rate indicates the number of requests from all performed requests that have succeeded. The results of the experiments are shown in figure 10. From the diagram it can be observed that the percentage of succeeded requests grows logarithmically with the number of nodes and degrades exponentially as load grows. The performance of eight nodes drops to 81% after receiving 1200 concurrent requests, however, 14 nodes can handle this load with 100% success rate. It can be also seen that with current test architecture adding more worker nodes does not show any visible improvement in the performance after 14 nodes in contrast when the setup was composed of 2, 4 and 6 nodes.

To sum it up, with current MCM implementation, two nodes deployed in different clouds can handle around 100-150 concurrent requests with 100% success rate. An addition of one node per cloud adds roughly the capacity of handling another 150 requests until the load grows up to 1800 concurrent requests, when the load balancer itself becomes a bottleneck. Hence adding more nodes does not improve the performance as desired. The analysis also shows that the elasticity of the cloud helps in achieving this required setup easily.

VIII. Conclusions

The paper introduced several cloud services and the inherent problems of using them for developing mobile applications. To address those problems, the paper proposed a generic middleware framework for providing hybrid mobile cloud services. The architecture of the MCM is explained in detail along with the asynchronous mechanisms, using third party push notification services (GCM/AC2DM and APNS) and the Mobile Host. The detailed analysis of the framework is provided with the study of two application scenarios, Zompopo and Bakabs. From the performance analysis of the applications, we could observe that the MCM shows reasonable performance levels of interaction with user, thus validating the proof of concept. To demonstrate

the horizontal scaling of the MCM, a scalability analysis is presented. The results show that MCM is horizontally scalable and can handle reasonable loads with significant ease.

While the prototype of MCM is working fine with the traditional web technologies like the HTTP and servlets, our future research will try to extend the architecture to better suit the cellular network, by providing the access to the MCM via the XMPP protocol. XMPP is an open technology for real-time communication, which powers a wide range of applications including instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware and generalized routing of XML data. However, this requires the protocol to be implemented for the mobile platforms we have considered in the analysis; Android and iOS. Another improvement involves the re-implementation of the middleware using Erlang [40] since it is more suitable for highly concurrent programs. Erlang also enables adding and replacing code while the system is running. Hence, the development of new MCM functionality can be done without restarting the entire system.

Another improvement involves extending the MCM to the Mobile Enterprise [32]. A Mobile Enterprise can be established in a cellular network by participating Mobile Hosts, which act as web service providers from smartphones, and their clients. Mobile Enterprise research built a Mobile Web Services Mediation Framework (MWSMF) based on the ESB technology that helps in handling QoS, discovery and integration issues of mobile web services. The main idea is to integrate MCM with MWSMF for providing a single interface for accessing both the mobile web services and mobile cloud services. The details will be addressed by our future publications.

IX. Acknowledgment

The research is supported by the European Social Fund through Mobilitas program, the European Regional Development Fund through the Estonian Centre of Excellence in Computer Science, the Estonian Science Foundation grant ETF9287 and the European Social Fund for Doctoral Studies and Internationalisation Programme DoRa.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., Above the clouds: A berkeley view of cloud computing, EECS Department, University of California, Berkeley, Tech.

[2] Apple Inc, IPhone, http://www.apple.com/iphone/.

[3] Google Inc, Android, http://www.android.com/.

[4] H. T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, Wireless communications and mobile computing 13 (18) (2013) 1587–1611.

[5] H. Flores, S. N. Srirama, C. Paniagua, A generic middleware framework for handling process intensive hybrid cloud services from mobiles, in: 9th Int. Conf. On Advances in Mobile Computing and Multimedia, ACM, 2011, pp. 87–94.

[6] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: Proceedings of the 8th international conference on Mobile systems, applications, and services, ACM, 2010, pp. 49–62.

[7] B. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in: Proceedings of the sixth conference on Computer systems, 2011, pp. 301–314.

[8] S. N. Srirama, H. Flores, C. Paniagua, Zompopo: Mobile Calendar Prediction based on Human Activities Recognition using the Accelerometer and Cloud Services, in: 5th Int. Conf. On Next Generation Mobile Applications, Services and Technologies (NGMAST), IEEE CS, 2011, pp. 63–69.

[9] S. Srirama, C. Paniagua, H. Flores, Social group formation with mobile cloud services, Service Oriented Computing and Applications 6 (4) (2012) 1–12.

[10] C. Paniagua, S. N. Srirama, 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, ACM, 2011, pp. 485–490.

[11] S. N. Srirama, M. Jarke, W. Prinz, Mobile web service provisioning, in: Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on, IEEE, 2006, pp. 120–120.

[12] Q. Wang, R. Deters, Soa's last mile connecting smartphones to the service cloud, in: 2009 IEEE International Conference on Cloud Computing, 2009, pp. 80–87.

[13] E. Cerami, S. Laurent, Web services essentials, O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2002.

[14] R. Aversa, B. Di Martino, M. Rak, S. Venticinque, Cloud agency: A mobile agent based cloud system, in: 2010 International Conference on Complex, Intelligent and Software Intensive Systems, Ieee, 2010, pp. 132–137.

[15] P. Narasimhan, Agora: mobile cloud-computing middleware, http://www.cylab.cmu.edu/research/projects/2010/.

[16] A. Weilenmann, C. Larsson, Local use and sharing of mobile phones, in: Wireless world, Springer, 2002, pp. 92–107.

[17] O. Davidyuk, J. Riekki, V.-M. Rautio, J. Sun, Context-aware middleware for mobile multimedia applications, in: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia, ACM, 2004, pp. 213–220.

[18] A. Hornsby, et al., From instant messaging to cloud computing, an xmpp review, in: 14th Int. Sym. On Consumer Electronics, IEEE, 2010, pp. 1–6.

[19] U. Hansmann, R. Mettala, A. Purakayastha, P. Thompson, SyncML: Synchronizing and managing your mobile data, Prentice Hall, 2003.

[20] A. Onetti, F. Capobianco, Open source and business model innovation. the funambol case, in: International Conference on OS Systems Genova, 11th-15th July, 2005, pp. 224–227.

[21] jets3t, jetS3t - An open source Java toolkit for Amazon S3 and CloudFront, http://jets3t.s3.amazonaws.com/toolkit/guide.html.

[22] Amazon, Inc, Amazon - Amazon Web Services, http://aws.amazon.com/.

[23] D. Nurmi, R. Wolski, C. G. G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The Eucalyptus Open-source Cloud-computing System, 2011.

[24] P. Saint-AndrÃl', K. Smith, R. Troncon, XMPP: the definitive guide : building real-time applications with Jabber, O'Reilly Media, 2009.

[25] JSON, JSON, http://www.json.org/.

[26] Facebook, Facebook - Mobile, http://www.facebook.com/mobile/.

[27] Face.com, Face.com, http://face.com/.

[28] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.

[29] Google, Inc, GCM - Google Cloud Messaging for Android, http://developer.android.com/guide/google/gcm/index.html.

[30] Apple, Inc, APNS, http://developer.apple.com/library/ios/.

[31] Google, Inc, AC2DM, http://code.google.com/android/c2dm/index.html.

[32] S. N. Srirama, M. Jarke, Mobile hosts in enterprise service integration, International Journal of Web Engineering and Technology (IJWET) 5 (2) (2009) 187–213.

[33] S. N. Srirama, M. Jarke, W. Prinz, Mobile web service provisioning, in: AICT-ICIW '06: Advanced Int. Conf. on Telecommunications and Int. Conf. on Internet and Web Applications and Services, IEEE Computer Society, 2006, p. 120.

[34] M. Owens, The definitive guide to SQLite, Apress, 2006.

[35] S. N. Srirama, O. Batrashev, E. Vainikko, SciCloud: Scientific Computing on the Cloud, in: The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGrid 2010), 2010, p. 579.

[36] N. Niclausse, A distributed load testing tool, http://tsung.erlang-projects.org/.

[37] Ateji Inc., Optimj - documentation, http://www.ateji.com/optimj/documentation.html.

[38] HTC Inc., HTC Inc, http://www.htc.com/www/.

[39] HAProxy, The Reliable, High Performance TCP/HTTP Load Balancer, http://haproxy.1wt.eu/.

[40] J. Armstrong, R. Virding, C. Wikstrom, M. Williams, Concurrent programming in erlang.